Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2009

Rui Miguel Teixeira de Sousa

Biblioteca para Comunicação entre FPGA e Dispositivos Periféricos

Universidade de Aveiro
2009

Departamento de Electrónica, Telecomunicações e Informática

Rui Miguel Teixeira de Sousa

Biblioteca para Comunicação entre FPGA e Dispositivos Periféricos

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. Valeri Skliarov, Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri

Presidente	Prof. Dr. António Manuel de Brito Ferrari Almeida professor catedrático da Universidade de Aveiro
Vogal – Orientador	Prof. Dr. Valeri Skliarov professor catedrático da Universidade de Aveiro
Vogal – Arguente Principal	Prof. Dr. Hélio Mendes de Sousa Mendonça professor auxiliar da Faculdade de Engenharia da Universidade do Porto

agradecimentos

Quero agradecer ao meu orientador, Prof. Dr. Valeri Skliarov, por todo o apoio dado para a elaboração deste trabalho, pelas ideias e conselhos, pelos meios disponibilizados e revisão desta dissertação.

Às duas pessoas que amplificaram grandemente o meu já existente interesse nas áreas dos sistemas reconfiguráveis, novamente o Prof. Dr. Valeri Skliarov e o Prof. Dr. Arnaldo Oliveira, pelo entusiasmo e gosto com que leccionam, pelos conhecimentos transmitidos e disponibilidade que sempre demonstraram.

Aos meus pais, Manuel e Maria Sousa, pela oportunidade que com grande esforço me concederam, pelo carinho e encorajamento, este marco do meu percurso é dedicado a eles.

À minha namorada, Filipa, pelo incentivo e pela compreensão da necessária gestão de tempo, agora já podemos passear todos os fins-de-semana que quiser.

palavras-chave

FPGA, núcleos IP, dispositivos periféricos, reutilização, parametrização, hardware template, repositório.

resumo

Esta dissertação apresenta um trabalho de desenvolvimento de metodologias e recursos para o aumento de produtividade no design de sistemas reconfiguráveis.
Os avanços explosivos na densidade de transístores por chip, permitem hoje em dia fabricar dispositivos de lógica programável de elevada capacidade como as FPGAs, suficiente para implementar sistemas inteiros de elevada complexidade, ainda assim os sistemas normalmente não são auto contidos e requerem interacção com componentes standard e dispositivos periféricos. Torna-se portanto necessário implementar as respectivas interfaces. O processo de design de sistemas reconfiguráveis através de linguagens HDL é muito semelhante ao desenvolvimento de software para computadores de uso geral, como tal, são aplicáveis algumas técnicas desse domínio, tais como a utilização de bibliotecas de funções, reutilização de código, construção hierárquica, macros e templates, estas técnicas tem como vantagens o aumento de produtividade e abstracção de complexidade.
O projecto é dedicado ao desenvolvimento dum conjunto de blocos reutilizáveis que implementam interfaces com periféricos de interacção com o utilizador, nomeadamente monitor VGA, teclado e rato, UART para ligação a PC com consola virtual/terminal de texto. Foi dedicada bastante ênfase na criação de parametrizações para os blocos desenvolvidos de modo a aumentar a adaptabilidade a diferentes alvos de integração, a criação de um hardware template como ponto de partida acelera o início de um novo projecto e a criação de um repositório on-line potencia a propagação e utilização do trabalho desenvolvido.

keywords

FPGA, IP cores, peripheral devices, design reuse, parameterization, hardware template, repository.

abstract

This dissertation presents a development work on methodologies and resources for the productivity increase in the design of reconfigurable systems. The explosive advances in the transistor density per chip allow nowadays to fabricate high capacity programmable logic devices such as FPGAs, sufficiently to implement entire systems with high complexity, nevertheless the systems normally are not self contained and require interaction with standard components and peripheral devices. It is therefore necessary to implement such interfaces. The design process of reconfigurable systems through HDL languages is very similar to the software development for general use computers, **as such**, some techniques from this domain are applicable, such as the use of function libraries, code reuse, hierarchical constructions, macros and templates, these techniques have the advantages of productivity increase and complexity abstraction.
The project is dedicated to the development of a set of reusable blocks that implement interfaces with peripherals for interaction with the user, namely VGA monitor, keyboard and mouse, UART for connection to a PC virtual console/text terminal. Much emphasis was dedicated in the creation of parameterizations for the developed blocks to increase the adaptability to different integration targets, the creation of a hardware template as a starting point accelerates the start of a new project and the creation of an on-line repository potentiates the promulgation and utilization of the developed work.

Conteúdo

Lista de Figuras

Lista de Tabelas

Convenções Tipográficas

Calibri	Corpo do texto
Calibri Itálico	Língua estrangeira (ex. *array*)
Cambria Bold	Rubricas das secções e subsecções
Cambria Itálico	Ênfase
Times New Roman Itálico	Unidades de grandezas (ex. *MHz*)
Tahoma	Nomes de ficheiros (ex. RAM.vhd)
Arial Narrow	Valores em base não decimal (ex. FAh ou 01b)
`Courier New`	Trechos de código VHDL, palavras reservadas, nomes de portos ou sinais (ex. `Clock`)

Capítulo 1

Introdução

"As ideias não são para guardar, alguma coisa tem de ser feita com elas."

Alfred North Whitehead, Matemático e Filósofo

Sumário

Neste capítulo encontra-se uma exposição do enquadramento da área de trabalho, apresenta-se a motivação que levou a esta proposta, delineiam-se os objectivos e a estrutura de organização deste documento.

1.1 Enquadramento

O projecto de sistemas digitais sofreu grandes evoluções nas últimas décadas. Ao sabor da progressão da lei de Moore [1], os sistemas digitais outrora constituídos com vários *chips* SSI (*Small-Scale Integration*) e MSI (*Medium-Scale Integration*), são actualmente implementados em *chips* de elevada densidade – VLSI (*Very Large-Scale Integration*). A possibilidade de integrar múltiplos e variados componentes num único substrato de silício permite tirar grandes vantagens da sua proximidade, tanto ao nível de tamanho como de desempenho e custo. Esta evolução permitiu que complexos projectos sejam implementados num único *chip* dando forma ao conceito SoC (*System-on-Chip*) onde um sistema computacional com processador, memórias, barramentos e periféricos é incorporado num só *chip*, indo mais além encontram-se os NoC (*Network-on-Chip*), são constituídos por vários sistemas que comunicam entre si assincronamente através de elementos *switches* ou *routers* como numa rede de comunicações convencional mas tudo no mesmo substrato de silício.

Dentro destes tipos de sistemas destacam-se duas metodologias de os concretizar, cada uma com as suas vantagens e desvantagens relativas, *não programável* – de funcionalidade definida durante o fabrico, ou *programável* – de funcionalidade definida após o fabrico e pelo utilizador.

A metodologia não programável permite características optimizadas, em área, consumo de potência e desempenho, os engenheiros desenvolvem o projecto articulando os blocos mais elementares, transístores → funções lógicas → *standard cells*, deste modo o tempo de projecto é normalmente demorado e os custos de produção e NRE (*Non-Recurring Engineering*) são elevados, requerem muitos meses de engenharia pré-produção e as máscaras litográficas para produção são extremamente caras e únicas a cada *chip*. Devido a estes factores, a utilização de ASICs (*Aplicattion-Specific Integrated Circuit*) só é economicamente viável para dispositivos destinados a mercados de grande dimensão, onde esses custos podem ser divididos pelo elevado número de componentes produzidos.

A metodologia programável recorre aos dispositivos PLD (*Programmable Logic Device*), existem duas categorias de PLDs de elevada capacidade, os CPLD (*Complex Programmable Logic Device*) e as FPGA (*Field-Programmable Gate Array*). Os CPLDs são constituídos por matrizes programáveis de portas lógicas que executam funções como soma de produtos e outros arranjos lógicos, linhas longas de ligação entre os blocos de lógica com pequenas matrizes de interligação, blocos de entrada/saída e alguma memória não-volátil. As FPGAs podem ser programáveis uma só vez ou reprogramáveis, são mais versáteis que os CPLDs porque os seus blocos de lógica programável são constituídos por LUTs (*Look-Up Tables*) ou tabelas de verdade, elementos combinatórios e sequenciais, blocos programáveis de entrada/saída, memória RAM (*Random Access Memory*) de duplo porto, multiplicadores dedicados, gestão de sinais de relógio e até microprocessadores (nos modelos mais avançados), linhas mais curtas e mais matrizes de interligação que permitem mais flexibilidade no encaminhamento das ligações. No entanto algumas capacidades são subaproveitadas, normalmente o projecto dum sistema não ocupa toda a capacidade da FPGA, os recursos de reprogramação também ocupam área adicional comparando com a implementação em ASIC aumentando o custo por unidade, as frequências de funcionamento são mais baixas, a potência dissipada tende a ser maior e a optimização possível é menor pois a posição dos blocos lógicos é fixa, o desempenho depende largamente da eficácia das ferramentas CAD (*Computer-Aided Design*) utilizadas.

A reprogramação, versatilidade e menor tempo de projecto fazem das FPGAs componentes muito atractivos para implementação de SoC e NoC, com a crescente complexidade dos sistemas, torna-se cada vez mais difícil desenvolver e comercializar um produto num curto *time-to-market*, sem que haja necessidade de actualizações posteriores para corrigir erros ou introduzir melhorias, através das FPGAs isso torna-se possível.

As FPGAs são hoje em dia aplicadas em inúmeras áreas distintas, indústria aeroespacial e automóvel, aplicações militares, processamento digital de sinal, prototipagem e emulação de sistemas e ASICs, criptografia, imagiologia médica, telecomunicações, bioinformática, co-processamento de algoritmos computacionalmente intensivos, a lista continua a crescer e crescer.

1.2 Motivação

Um modo de diminuir o tempo de projecto é a utilização de níveis hierárquicos de abstracção e reutilização de módulos pré-desenvolvidos, esta metodologia aplicada a outras áreas também é aplicável ao projecto de sistemas em FPGA.

A utilização de bibliotecas de componentes e/ou módulos é por isso desejável e extremamente útil.

No Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, são leccionadas diversas disciplinas e há projectos de investigação em curso que requerem o uso de placas de desenvolvimento com FPGA, quer para desenvolvimento directamente aplicável a estas, quer como base de suporte a conteúdos programáticos menos direccionados a FPGAs.

A inexistência, na Universidade de Aveiro, de uma biblioteca deste género é a principal motivação que levou a esta proposta de dissertação e ao trabalho por mim realizado.

1.3 Objectivos

1) O trabalho subjacente a esta dissertação visa a criação de uma biblioteca de módulos para interfaces comummente presentes em placas de desenvolvimento/prototipagem cujo elemento principal seja uma FPGA.

Os módulos propostos são principalmente de interacção com o utilizador, nomeadamente:

- interface com monitor VGA (*Video Graphics Array*)
- geração de caracteres e visualização de figuras gráficas
- interface com teclado PS/2 (IBM *Personal System/2*)
- interface com rato PS/2
- interface de comunicações série RS-232

2) Criação de um *Web site* que funcione como repositório para os módulos desenvolvidos e sua documentação descritiva e de utilização.

3) Validação dos módulos desenvolvidos através de aplicação de demonstração.

Seleccionou-se a implementação de um algoritmo D* Lite (*D-star*, por extenso) de navegação em terreno apenas parcialmente conhecido, uma variante mais avançada deste algoritmo é utilizada pelos robôs autónomos *Spirit* e *Opportunity* em Marte.

1.4 Organização da dissertação

O capítulo 2 desenvolve-se sobre a arquitectura de FPGAs, tecnologias de fabrico e seus blocos constituintes.

No capítulo 3 é efectuada uma comparação deste trabalho com outros trabalhos de natureza similar, realçando as potencialidades deste trabalho sobre os demais, são abordados os aspectos técnicos dos equipamentos com que se pretende implementar as interfaces e explicados todos os módulos desenvolvidos do ponto de vista das funcionalidades implementadas.

O quarto capítulo indica a funcionalidade do *hardware template* criado para a fácil utilização e configuração modular das interfaces desenvolvidas, bem como a sua utilização.

O capítulo 5 fecha com a conclusão e apresenta ideias para trabalho futuro na linha de desenvolvimento desta proposta de trabalho.

No capítulo 6 são integrados os apêndices, num dos quais se encontra a lista de acrónimos.

O capítulo 7 apresenta a bibliografia de conteúdos referenciados.

Capítulo 2

FPGAs e o seu fluxo de projecto

"Com uma única patente, Ross incendiou o espírito de inovação que construiu uma indústria."

Moshe Gavrielov, Presidente e CEO da Xilinx Inc.

Sumário

Neste capítulo desenvolve-se sobre a arquitectura de FPGAs, tecnologias de fabrico e seus blocos constituintes.

2.1 Introdução

Os dispositivos FPGA foram inventados em 1984 por Ross Freeman, co-fundador da Xilinx Inc., a principal diferença com os PLDs existentes na altura residia na flexibilidade de interligação dos blocos lógicos. Com o crescimento da indústria dos dispositivos lógicos reconfiguráveis, naturalmente surgiram competidores de mercado, como a Altera Corp., a Actel, Lattice Semiconductor e Atmel entre outros, sendo a liderança ocupada pela Xilinx Inc. com 51% de mercado seguida pela Altera Corp. com 34% em 2007 [2].

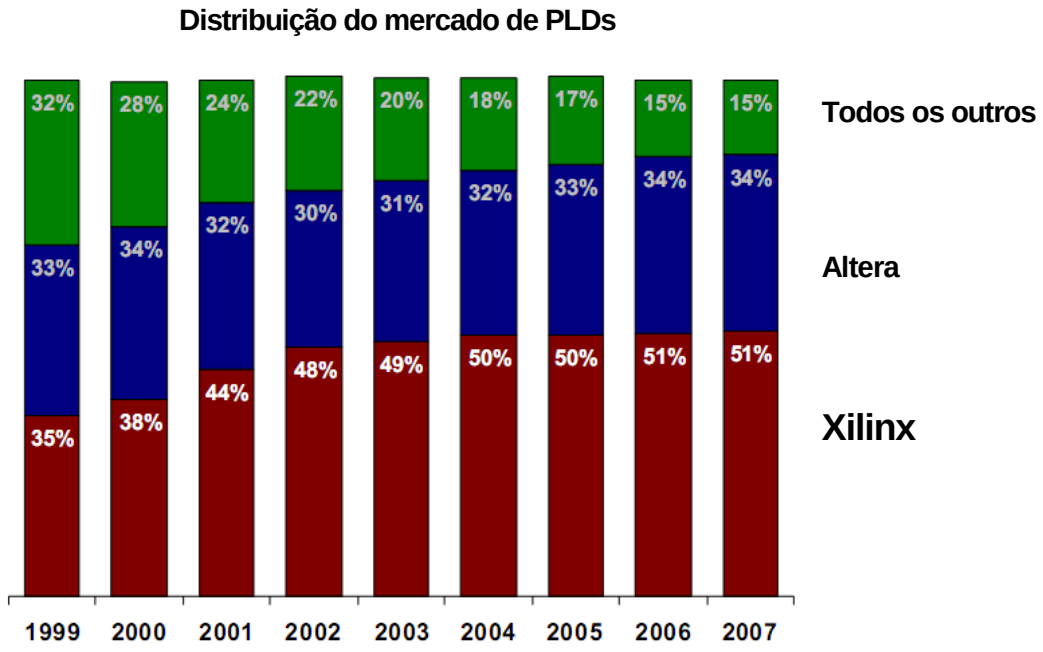


Figura 2.1 – Distribuição do mercado de PLDs pelos principais fabricantes. [2]

A seguinte descrição dos blocos fundamentais que constituem uma FPGA reflecte principalmente a arquitectura adoptada pela Xilinx Inc., embora sejam mencionados aspectos presentes noutras topologias adoptadas pelos restantes fabricantes a termo de comparação.

Os módulos foram desenvolvidos com as FPGAs Spartan–3 da Xilinx em mente, esta escolha assenta sobre dois factores determinantes, em primeiro a Xilinx Inc. é o líder de mercado em vendas de FPGAs, em segundo, as placas de prototipagem e desenvolvimento disponíveis no DETI (Departamento de Electrónica, Telecomunicações e Informática) incorporam FPGAs deste fabricante e maioritariamente da família Spartan–3.

2.2 Arquitectura das FPGAs

Numa abordagem muito simplificada, uma FPGA é constituída por um *array* bidimensional ou matriz de blocos de lógica programável e abundantes ligações programáveis em redor destes, permitindo assim múltiplos caminhos disponíveis que podem ligar os blocos uns aos outros. Existem também na periferia do *chip* blocos que regulam a entrada e/ou saída de sinais para o exterior da FPGA.

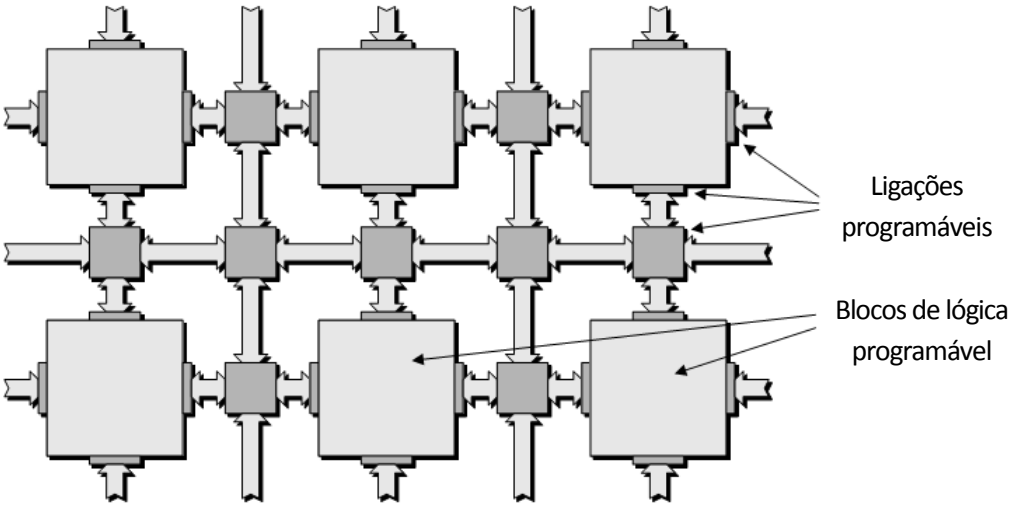


Figura 2.2 – Estrutura básica de uma FPGA. [3]

Este tipo de disposição da Figura 2.2 é usada pela Xilinx [3] e é normalmente designada de *island-style* ou ilhas, uma vez que cada bloco de lógica programável está rodeado de ligações por todos os lados.

Existem outros tipos de disposição, em linha ou *row-based* em que os recursos de interligação estão distribuídos principalmente em linhas assim como os blocos de lógica programáveis, usada pela Actel [4] [5].

Outro tipo de disposição é a hierárquica usada pela Altera Corp. [6] [5], existem dois níveis de hierarquia de ligações principais, os blocos de lógica encontram-se associados espacialmente em vários grupos, um nível conecta blocos lógicos entre si e o nível seguinte conecta os vários grupos entre si.

Uma FPGA não é apenas constituída por blocos de lógica programável e ligações programáveis, com o avanço tecnológico a permitir uma cada vez maior densidade de transístores por unidade de área e com a disseminação das FPGAs pelas mais variadas aplicações, tornou-se necessário integrar na arquitectura outros blocos mais especializados de modo a aumentar a performance e satisfazer novas necessidades.

É de extrema importância para o projectista ter um conhecimento sólido de como funcionam estes blocos básicos, de modo a escolher a FPGA mais adequada à aplicação e posteriormente à utilização eficiente de recursos no desenvolvimento do projecto.

2.2.1 Blocos de lógica programável

Os Blocos de lógica programável ou CLBs (*Configurable Logic Blocks*)[*] são os principais elementos usados nos projectos em FPGA, são eles que permitem a maior parte das funcionalidades possíveis de implementar. Um CLB é constituído por blocos mais elementares como demonstrado na Figura 2.3.

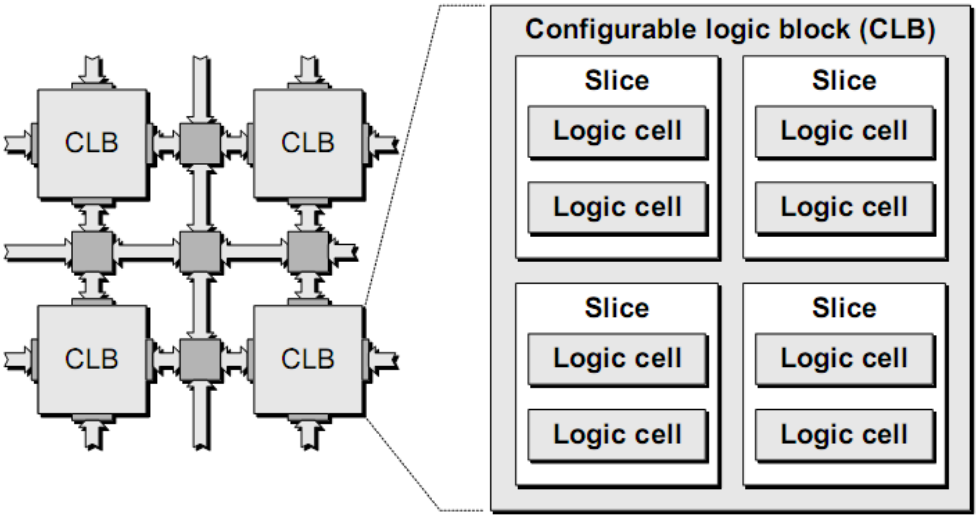


Figura 2.3 – Decomposição de um CLB em blocos mais elementares. [3]

De acordo com a arquitectura da FPGA Spartan-3, cada CLB é composto por quatro *slices* e cada uma contém duas *logic cells*. A organização das *slices* dentro de um CLB é detalhada na Figura 2.4.

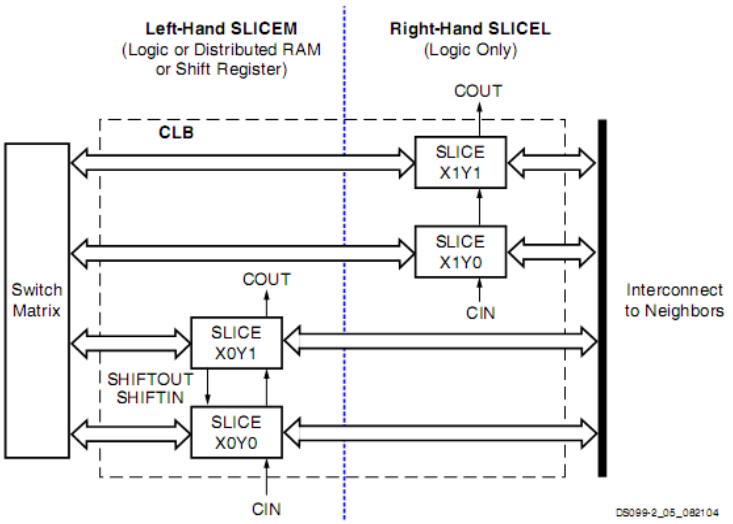


Figura 2.4 – Organização das *slices* num CLB. [7]

[*] A Xilinx usa a terminologia CLB (*Configurable Logic Block*) enquanto a Altera usa LAB (*Logic Array Block*), ambos referem-se ao mesmo princípio de bloco de lógica programável com algumas diferenças estruturais entre si.

Uma *logic cell* é composta por uma LUT, um registo tipo flip-flop D, multiplexadores, algumas portas lógicas elementares e permite propagação de *carry* de umas células para as outras. As *slices* da esquerda na Figura 2.4 para além de LUTs de quatro entradas também podem ser configuradas como *shift registers* de 16 *bits* ou memórias de 16×1 *bits* designadas *distributed RAM* pois os CLBs estão distribuídos por toda a FPGA.

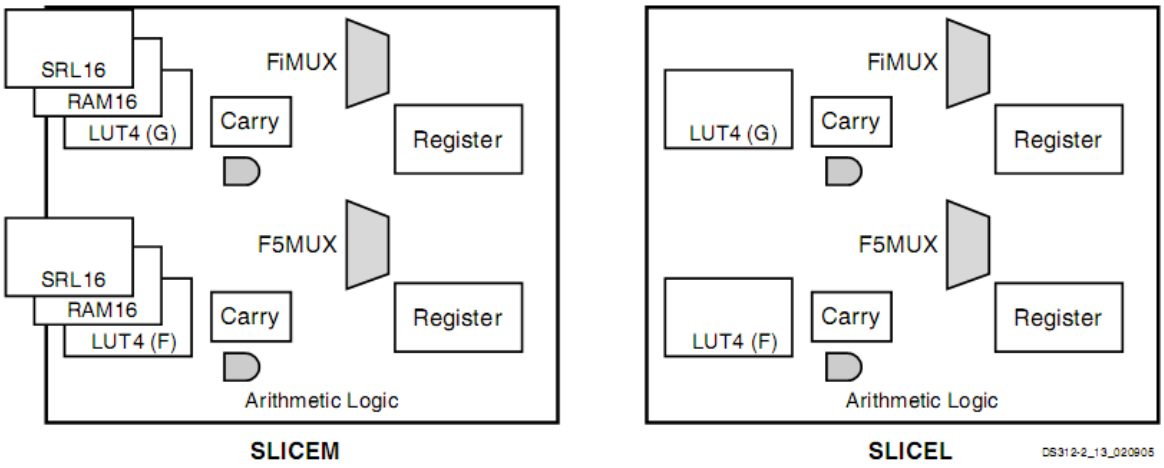


Figura 2.5 – Estrutura simplificada das *slices*. [7]

Relativamente aos conteúdos dos seus blocos de lógica programável, as FPGAs podem ser categorizadas em termos de *granulosidade*:

- *fina* – implementam funções lógicas elementares de poucas entradas ou um elemento de memória flip-flop D ou uma *latch* D. Deste modo é possível uma grande utilização de todos os recursos de um bloco, no entanto como os blocos são simples, requerem mais interligações e maior quantidade de memória que mantém a configuração.
- *média* – implementam funções lógicas de mais entradas (com LUTs tipicamente de 4 a 6 entradas), elementos de memória flip-flop D ou *latches* D, multiplexadores e possuem lógica elementar para realização de operações aritméticas e propagação de *carry*. Permitem uma optimização melhor da área total da FPGA para funções mais complexas, no entanto a realização de funções simples implica que haja recursos subaproveitados. As FPGAs da Xilinx encaixam nesta categoria.
- *grossa* – implementam um misto de granulosidade média com funções de complexidade mais elevada ao nível algorítmico ou unidades de processamento dentro dos blocos, acentuam tanto os benefícios como os problemas em comparação à granulosidade média.

Este é o panorama geral de organização de um CLB hoje em dia, no entanto… e citando directamente do livro [8]:

"The definition of what forms a CLB varies from year to year. In the early days, a CLB consisted of two 3-input LUTs and one register. Later versions sported two 4-input LUTs and two registers. Now, each CLB can contain two or four slices, where each slice contains two 4-input LUTS and two registers. And as for the morrow well, it would take a braver man than I even to dream of speculating." – Clive Maxfield.

2.2.2 Recursos de encaminhamento e ligações programáveis

Os recursos de encaminhamento e as suas ligações ocupam normalmente uma área superior aos CLBs, as linhas de ligação estendem-se por toda a FPGA ao redor dos CLB e restantes blocos, de modo a providenciarem ligações eficientes existe toda uma rede de segmentos de diferentes comprimentos, nas extremidades encontram-se as ligações programáveis.

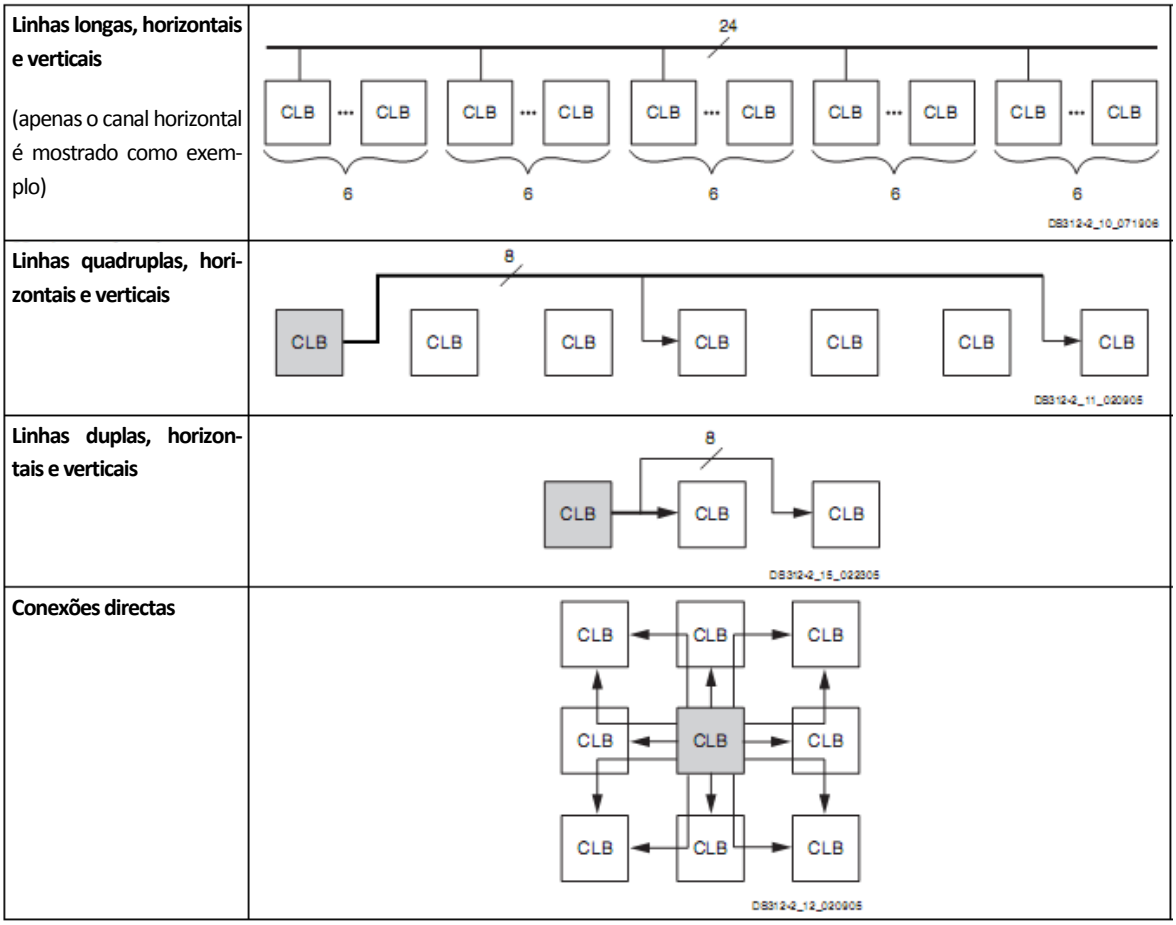


Figura 2.6 – Tipos de linhas de encaminhamento entre CLBs na FPGA Spartan-3. [7]

Consoante a tecnologia usada pelo fabricante, as ligações programáveis podem ser de vários tipos:

2.2.2.1 SRAM (Static Random Access Memory)

Este é o tipo de configuração mais utilizado, uma célula de memória de 1 *bit* mantém um transístor de passagem ou um *buffer tri-state* num estado de passagem ou de bloqueio, permite a rápida (re)configuração da FPGA já no protótipo ou produto final. São fabricadas em CMOS (*Complementary Metal-Oxide Semiconductor*) e devido à investigação e desenvolvimento impulsionados pelas companhias fabricantes de memórias para servirem o mercado de computadores pessoais, esta é uma tecnologia que está no seu auge de maturidade pelo que os processos de fabrico estão bem consolidados. Tem contudo a desvantagem de ser um tipo de memória volátil, cada vez que a FPGA é desligada da alimentação a sua

configuração perde-se, na utilização seguinte a programação tem de ser novamente descarregada de uma memória não-volátil exterior, de um microcontrolador ou de um computador pessoal. Tem também a desvantagem de uma célula SRAM necessitar de mais área que outras tecnologias, cada célula é composta por 6 transístores e consome potência estática, i.e. na manutenção do *bit* armazenado.

2.2.2.2 Anti–fusível

Ao contrário das FPGAs baseadas em SRAM, as FPGAs de anti-fusível tem de ser programadas fora do sistema num programador próprio, não são possíveis de reprogramar uma vez que o anti-fusível é uma ligação permanente, pequenas secções de silício amorfo (não cristalino e não condutor) são selectivamente fundidas durante a programação transformando essas secções em polisilício condutor, isto torna-as não-voláteis, assim que se liga o sistema, a FPGA já está configurada. Uma outra vantagem é a capacidade das ligações em anti-fusível dotarem a FPGA de uma certa resistência à radiação, isto é de particular interesse nas aplicações militares e aeroespaciais no entanto não é suficiente, para essas aplicações é necessária protecção extra e incorporar redundância espacial e temporal no projecto. As FPGAs de anti-fusível também requerem mais passos no fabrico que as de SRAM, deste modo o processo não está tão bem desenvolvido em relação ao *scaling* dos transístores que são de maiores dimensões. Relativamente ao consumo de potência, este é muito baixo devido ao polisilício condutor ter muito baixa resistência.

2.2.2.3 Flash e E²PROM (ou EEPROM – Electrically Erasable Programmable Read Only Memory)

As células de memória E^2PROM e Flash são compostas por dois transístores sendo um deles de *gate* flutuante que armazena o *bit* e o segundo que é usado para apagar o *bit*, o que as torna mais pequenas que as células SRAM, permitindo que o resto da lógica esteja mais compacta reduzindo os atrasos de propagação. Estes são tipos de memória não-volátil e cuja programação pode ser feita num programador próprio ou já em sistema. Combinam portanto as características de reprogramabilidade da SRAM e não-volatilidade do anti-fusível. São no entanto cerca de três vezes mais lentas a gravar que as células SRAM pois necessitam do apagamento prévio à escrita, requerem cinco passos adicionais no fabrico relativamente ao processo CMOS. As células Flash tem o isolamento da *gate* mais fino que as E^2PROM o que as torna ligeiramente mais rápidas, existem também implementações de um único transístor de *gate* flutuante mas nesse caso a escrita é precedida do apagamento de todas as células pois o transístor auxiliar não se encontra presente, outra inovação nas células Flash é a capacidade de poder armazenar dois *bits* através da quantificação da carga a manter na *gate* flutuante. [8]

2.2.2.4 Segurança da Propriedade Intelectual

Uma consideração a ter relativamente à propriedade intelectual – IP (*Intellectual Property*) é a segurança, as empresas que desenvolvem sistemas obviamente não querem ver copiados os seus produtos e funcionalidades, quer o desenvolvimento seja feito internamente ou por empresas externas que vendem núcleos IP modulares para os primeiros integrarem nos seus produtos.

Como a configuração das FPGAs de tecnologia SRAM é armazenada externamente e carregada a cada utilização, com o equipamento adequado torna-se fácil a leitura dessa configuração e portanto a replicação do projecto.

Já existem em grande variedade FPGAs que contém um registo que guarda uma chave de encriptação fornecida pelos engenheiros de projecto e que impossibilita a leitura da configuração após a programação, deste modo a configuração é encriptada e gravada na memória exterior de suporte, quando a FPGA é ligada à alimentação a configuração é descarregada e desencriptada já dentro da FPGA o que torna a violação da propriedade intelectual bem mais difícil ou mesmo impossível, o registo que possui a chave é mantido por uma pilha que pode durar vários anos, no caso de a pilha ser retirada ou perder a carga, o equipamento perde a chave de encriptação e a configuração já não pode ser utilizada.

As FPGAs de anti-fusível são impossíveis de replicar, são activados anti-fusíveis que impossibilitam a leitura a partir do exterior após a programação, durante a programação o dispositivo pode ser lido pelo aparelho programador de modo a assegurar que esta é executada sem erros.

Relativamente às FPGAs de Flash e E^2PROM, também é gravada uma chave multi-bit para encriptação, como estes dispositivos são não-voláteis, não requerem o uso de pilha para manter a chave, reprogramações posteriores à configuração da chave são possíveis apenas pelo porto JTAG (*Joint Test Action Group*) e com fornecimento da chave de encriptação.

Tabela 2.1 – Resumo das características mais relevantes nas tecnologias de ligações programáveis.

	SRAM	Anti-fusível	Flash/E^2PROM
Maturidade tecnológica	Bem consolidada	Uma ou mais gerações atrás	Uma ou mais gerações atrás
Reprogramável	Sim (em sistema)	Não	Sim (em sistema ou offline)
Velocidade de reprogramação (inclui apagamento)	Rápida	---	1/3 da SRAM
Volátil	Sim	Não	Não
Necessita configuração externa	Sim	Não	Não
Bom para prototipagem	Sim (muito bom)	Não	Razoável
Ligação instantânea	Não	Sim	Sim
Segurança IP	Aceitável (requer encriptação de bitstream)	Muito bom	Muito bom
Tamanho da célula de configuração	Grande (6 transístores)	Muito pequena	Média-pequena (2 transístores)
Consumo de potência	Médio	Baixo	Médio
Protecção contra radiação	Não	Sim	Não

2.2.3 Blocos de Entrada/Saída

Os blocos de entrada/saída ou IOBs (*Input/Output Blocks*) são a interface entre os pinos do encapsulamento e a lógica interior da FPGA, são também eles recursos programáveis na medida em que providenciam um controlo de fluxo unidireccional ou bidireccional e um suporte de uma ampla gama de interfaces estandardizadas, como referência, a FPGA Spartan-3 permite vários modos de utilização diferencial e de pino único com configuração da potência e *slew-rate* das saídas, entradas e saídas sequenciais e combinatórias com registos DDR (*Double Data Rate*) para ligação directa com memórias DDR, atrasos programáveis nas entradas, controlo de terminação/acoplamento resistivo para minimizar ou eliminar reflexões de sinal e resistências de *pull-up* e *pull-down*. [7]

2.2.4 Memória RAM embutida

Outro elemento essencial encontrado em FPGAs é a memória RAM embutida, muitas aplicações requerem a utilização de memória, deste modo as FPGAs permitem satisfazer essa necessidade com blocos de memória dentro do *chip*, de modo a reduzir os atrasos e custos da utilização de memórias exteriores.

Podem ser encontradas relativamente dispersas pela FPGA ou agrupadas em colunas consoante o fabricante.

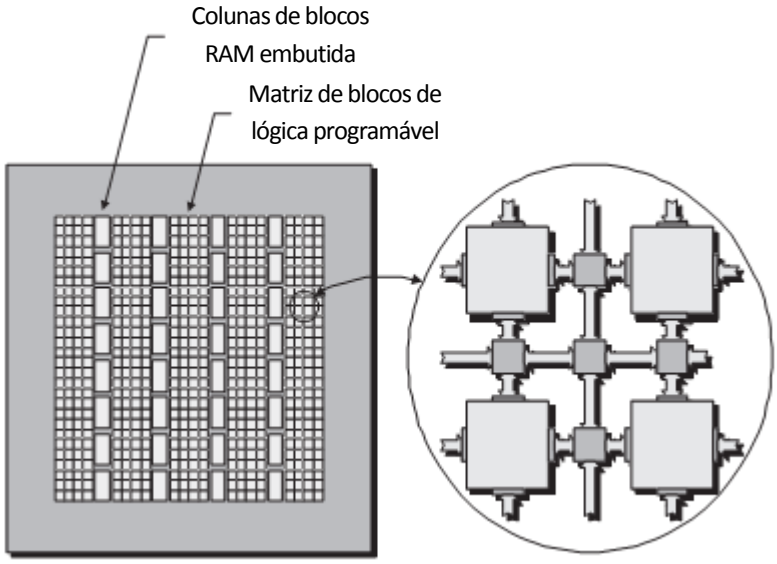


Figura 2.7 – Exemplo de distribuição de memória RAM embutida numa FPGA. [3]

Estas memórias também podem ser configuradas para um ou dois portos de acesso e utilizadas bloco a bloco ou agrupadas para obter mais capacidade e/ou largura das palavras de *bits*. Outras aplicações estendem-se à utilização como ROMs (*Read-Only Memory*), FIFOs (*First-In First-out*), LUTs ou máquinas de estados de grande dimensão, *buffers* circulares e *shift-registers*, entre outras.

2.2.5 Multiplicadores, somadores e MACs (Multiply-and-Accumulate)

Encontram-se também blocos especializados para funções aritméticas o que torna estas operações muito mais rápidas do que com CLBs, são muitas vezes usados em conjunto com as memórias RAM embutidas estando normalmente na sua proximidade.

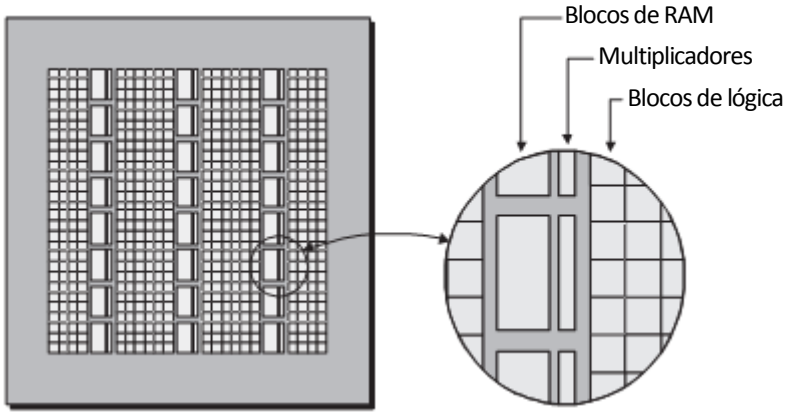


Figura 2.8 – Blocos multiplicadores dedicados junto às memórias RAM embutidas. [3]

Como as FPGAs também são usadas em aplicações de processamento digital de sinal – DSP (*Digital Signal Processing*), uma função muito usada em DSP é a multiplicação e acumulação do resultado, como tal já é habitual encontrar também blocos dedicados que implementam esta função – MACs ou blocos DSP.

2.2.6 Distribuição e gestão de sinais de relógio

Todos os elementos síncronos na FPGA funcionam ao ritmo de um sinal de relógio exterior, a distribuição deste sinal é feita por uma rede de ligações dedicadas de modo a chegarem a todos os CLBs com o mínimo desfasamento temporal (ou *skew*) entre si. O número destas redes é muito limitado.

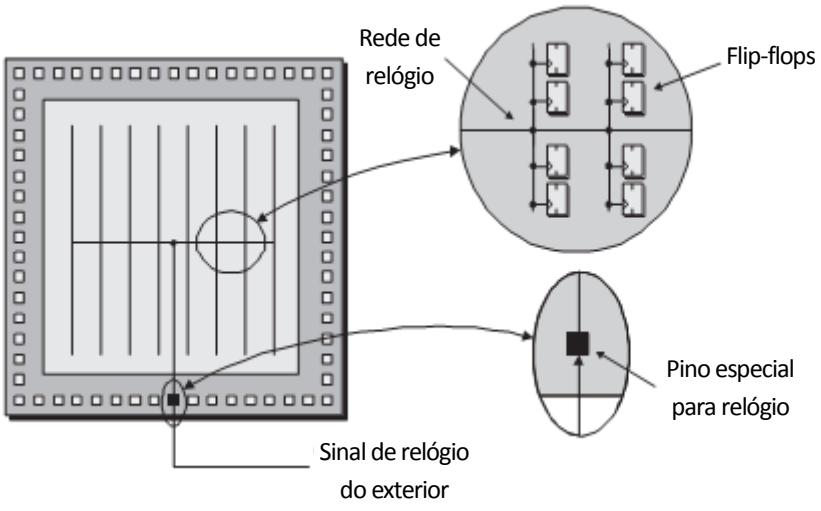


Figura 2.9 – Rede muito simplificada de distribuição de relógio. [3]

Existem também alguns (poucos) blocos gestores de relógio – DCM (*Digital Clock Manager*) que permitem mais algumas funcionalidades muito úteis, como a síntese de relógios com diferentes frequências através da multiplicação e/ou divisão do relógio exterior, eliminação de *jitter* e de *skew*, desfasamento relativo entre vários relógios através de atrasos programados, é possível portanto obter vários domínios de relógio com diferentes frequências e/ou fases relativas, a aplicação de vários relógios num só projecto deve ser cuidada para não obter problemas de meta-estabilidade devido à violação dos tempos de *setup* e *hold* nos elementos síncronos.

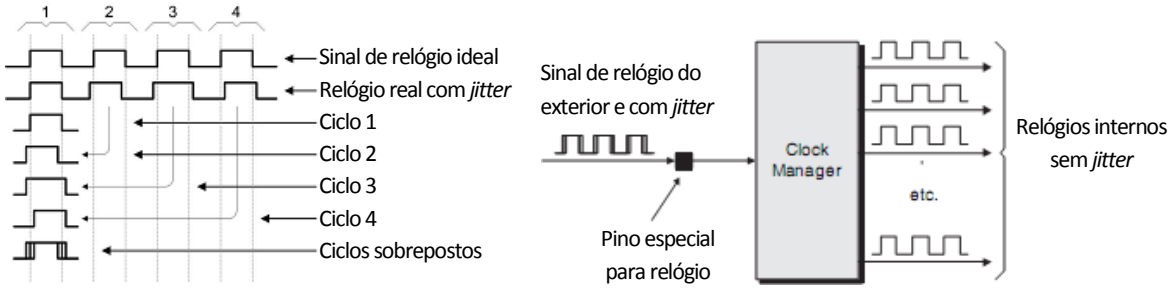


Figura 2.10 – Utilização do bloco DCM na remoção de *jitter*. [3]

2.2.7 Processadores embutidos

É normal que os projectos de sistemas electrónicos contenham um microprocessador ou microcontrolador, mesmo em sistemas combinados com FPGAs, faz sentido que estes microprocessadores sejam portados para dentro das FPGAs, quer para reduzir o custo de um dispositivo extra quer na redução espaço ocupado e complexidade das placas de circuito impresso – PCBs (*Printed Circuit Boards*).

Os fabricantes de FPGAs oferecem alguns núcleos IP de microprocessadores (*soft microprocessor cores*) que são implementados na lógica de uso geral da FPGA, como por exemplo o MicroBlaze da Xilinx ou o NIOS 2 da Altera, existem também modelos mais avançados de FPGA que contém um microprocessador (ou vários, tipicamente 1, 2 ou 4) construído no substrato de silício (*hard microprocessor core*) juntamente com os restantes blocos, como os PowerPC das famílias Virtex da Xilinx ou os processadores ARM embutidos em dispositivos da Altera. Embora os processadores *hard* tenham uma performance superior aos *soft*, estes últimos oferecem mais flexibilidade através de parametrização e permitem um número variável de instanciações para multi-processamento ao contrário dos *hard* que são fixos.

2.2.8 Transmissores/Receptores de alto débito

Para além das várias interfaces estandardizadas disponíveis nos IOBs, também é já usual encontrar FPGAs com blocos dedicados para transmissão e recepção de sinais de alto débito sobre diferentes interfaces como por exemplo PCI-E (*Peripheral Component Interconnect - Express*), SATA (*Serial Advanced Technology Attachment*) ou Gigabit Ethernet entre outros. Estes blocos dedicados permitem manter a integridade de sinal a altas frequências de comunicação.

2.3 Fluxo de projecto

Implementar um projecto em FPGA requer a programação de milhares de ligações programáveis e inicialização de valores lógicos, uma tarefa impossível de realizar sem a ajuda de ferramentas CAD, desde a especificação funcional do circuito até à geração do ficheiro de programação (*bit-stream*) há uma série de passos e procedimentos que devem ser executados.

2.3.1 Especificação

A especificação é efectuada através de linguagens de descrição de *hardware* – HDL (*Hardware Description Language*), destacam-se VHDL (*Very High Speed Integrated Circuit HDL*) e Verilog que são de médio-baixo nível, SystemC e HandelC que são linguagens de alto nível. É possível também, dependendo dos programas proprietários dos fabricantes de FPGAs, efectuar a especificação do sistema (ou partes deste) através de esquemas de circuitos, diagramas de fluxo de sinal ou diagramas de estados. É nesta fase do projecto que os núcleos IP são utilizados permitindo aumentar o nível de abstracção e aumentando significativamente a produtividade.

Através de programas de simulação que traduzem a especificação em formas de onda, efectua-se uma análise funcional preliminar que permite detectar e corrigir erros de funcionalidade, nesta fase de simulação não são tidos ainda em conta os tempos de propagação dos sinais.

2.3.2 Síntese

A síntese corresponde à transformação da especificação do sistema feita pelo projectista, numa especificação feita pelo *software* em termos de funções ou estruturas lógicas e suas dependências, este resultado a que se chama *net-list* pode ser optimizado pelo programa do fabricante ou por outros programas com algoritmos mais eficientes, uma vez que este passo é independente da tecnologia específica do fabricante, família de FPGAs ou modelo.

2.3.3 Mapeamento

Este paço e os seguintes já são dependentes da tecnologia de cada FPGA, os elementos da *net-list* são mapeados e optimizados para primitivas específicas a cada modelo, como LUTs, registos, etc., que são mapeados em CLBs, BLock RAMs que são agrupadas de modo a constituir memórias de maior capacidade, multiplicadores e outras estruturas. O resultado é uma nova *net-list* com um nível de maior abstracção e especifica a cada FPGA.

2.3.4 Posicionamento & encaminhamento

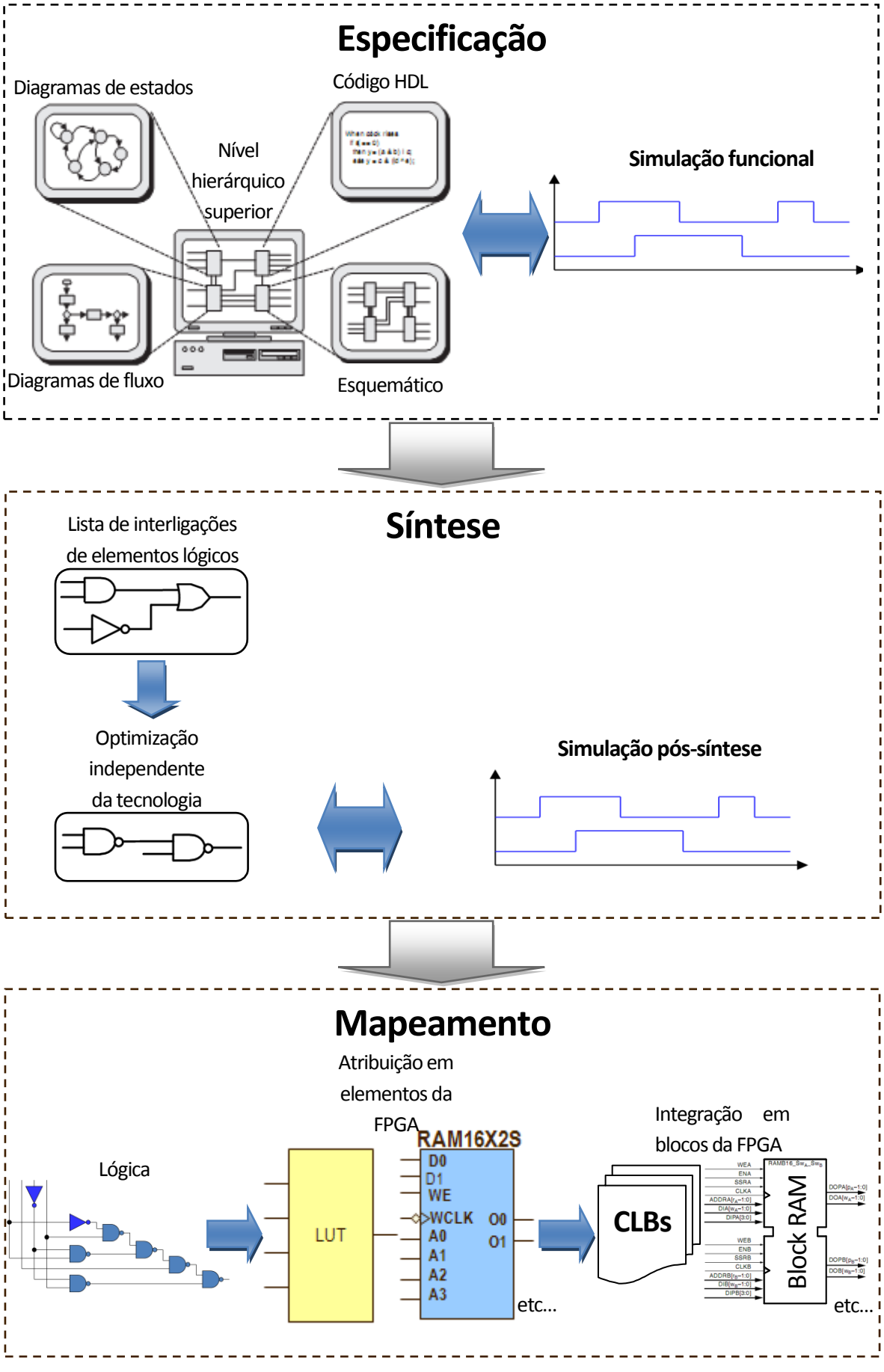
Nesta fase os blocos alocados no mapeamento vão ser distribuídos por localizações na FPGA e escolhidas as melhores ligações entre blocos, de modo a obter a melhor performance respeitando eventuais restrições impostas pelo projectista. Com os novos dados sobre a localização e ligações usadas é possível obter uma simulação de performance com atributos temporais relativos aos atrasos de propagação. Podem seguir-se rondas de optimização em que os blocos mais intensamente usados são reposicionados (*floor-planning*), quando mesmo assim não é possível obter a performance necessária, pode recorrer-se a um modelo de FPGA com melhor desempenho ou uma nova especificação mais optimizada se possível.

2.3.5 Programação e depuração em sistema

Se após as fases anteriores resultarem num projecto satisfatório, é gerado o ficheiro de programação, este pode ser descarregado para a FPGA através do porto JTAG (também conhecido por norma *IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture*). Este porto para além de permitir a programação, permite a leitura de sinais no interior da FPGA recorrendo a alguns CLBs que estejam disponíveis e alguma RAM que são integrados no projecto como pontos de teste, os dados recolhidos dos sinais que se querem monitorizar podem ser vistos em programas de PC para esse efeito, é também possível usar equipamento de teste como osciloscópios e analisadores lógicos, no entanto são necessários pontos de teste nas PCBs e as próprias pontas de sinal podem causar distúrbios na integridade do sinal, além disso é completamente impossível aceder aos sinais interiores da FPGA que não sejam encaminhados para os pinos do encapsulamento.

Existem normalmente ligações manuais (*jumpers* ou interruptores) que direccionam a programação directamente para a FPGA ou para a memória de suporte (nas FPGAs SRAM) ou ambos, é frequente encontrar também em placas de prototipagem ou de teste/avaliação, outras interfaces que permitem a programação como por exemplo USB (*Universal Serial Bus*) ou porta paralela.

Encontra-se nas páginas seguintes um resumo em forma de diagrama das diversas fases do fluxo de projecto em FPGAs (Figura 2.11 – Diagrama do fluxo de projecto em FPGA).


Especificação
Diagramas de estados
Código HDL
Nível hierárquico superior
Simulação funcional
Diagramas de fluxo
Esquemático
Síntese
Lista de interligações de elementos lógicos
Optimização independente da tecnologia
Simulação pós-síntese
Mapeamento
Atribuição em elementos da FPGA
Integração em blocos da FPGA
Lógica
LUT
RAM16X2S
D0
D1
WE
WCLK
A0
A1
A2
A3
O0
O1
etc...
CLBs
Block RAM
etc...

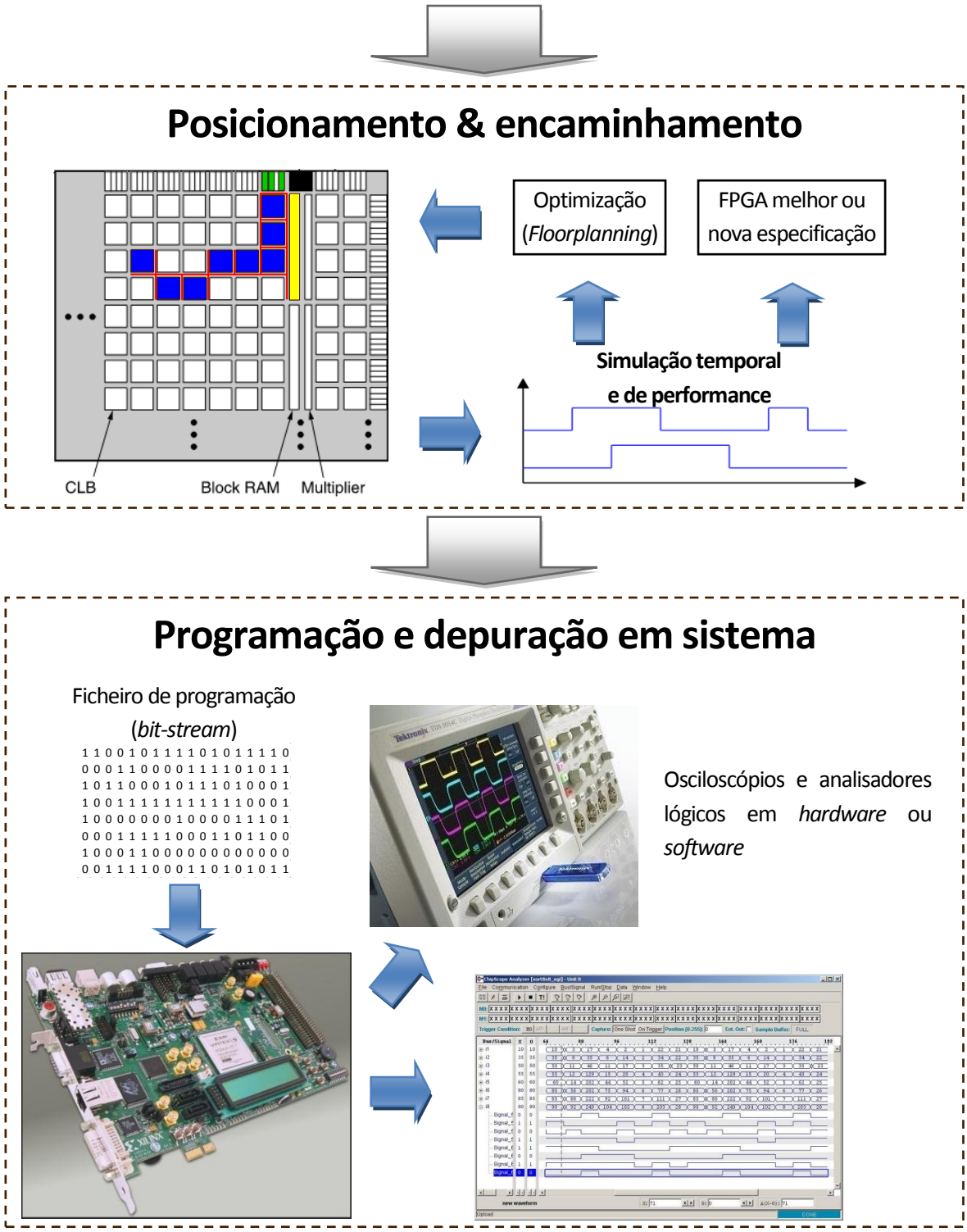


Figura 2.11 – Diagrama do fluxo de projecto em FPGA

Capítulo 3

Módulos reutilizáveis desenvolvidos

Sumário

Neste capítulo é efectuada uma comparação deste trabalho com outros trabalhos de natureza similar, realçando as potencialidades deste trabalho sobre os demais, são abordados os aspectos técnicos dos equipamentos com que se pretende implementar as interfaces e explicados todos os módulos desenvolvidos do ponto de vista das funcionalidades implementadas.

3.1 Introdução

Como referido anteriormente, o trabalho executado para esta dissertação assenta no desenvolvimento de módulos reutilizáveis para interfaces presentes em placas de prototipagem/desenvolvimento com FPGAs.

Trata-se de interfaces bastante vulgares neste tipo de placas, no entanto a disponibilidade de controladores reduz-se a alguns núcleos IP oferecidos em conjunto com a compra de programas comerciais, neste caso o código não é aberto ao utilizador, geralmente a interface corresponde à ligação do núcleo a um barramento de processador (quer seja *soft* ou *hard*) como o IBM *CoreConnect* [9] para os PowerPC ou o *Wishbone bus* [10] que é *open-source*, um dos mais populares é o AMBA [11] [12] (*Advanced Microcontroller Bus Architecture*) desenvolvido pela ARM (*Advanced RISC Machines, Ltd.*), estas soluções são mais indicadas para o co-projecto de *hardware* e *software* do que para *hardware* unicamente.

Existem também algumas opções de código aberto que dependem da boa vontade e experiência dos projectistas, quer sejam *hobbyists*, estudantes ou engenheiros experientes, também estes recursos são pouco sólidos pois normalmente foram feitos para um projecto em particular e oferecem pouca flexibilidade ou não se ajustam às necessidades.

Alguns livros também contêm capítulos sobre alguns dos módulos propostos, mas apenas numa perspectiva introdutória e incompleta, deixando a implementação totalmente funcional como exercício a cargo do leitor [13] [14].

Muitas vezes os projectistas vêem-se na situação de ter de "reinventar a roda" para prosseguirem os seus trabalhos, por exemplo no artigo [15] que refere experiências no ensino duma cadeira de sistemas embutidos baseada em FPGA, o autor refere vários projectos com interacção através do monitor em que se desenvolviam os controladores à medida, embora esta abordagem seja útil na aquisição de conhecimentos e metodologias para projectos muito simples para ensino, a um nível mais complexo como investigação e desenvolvimento torna-se numa perda de produtividade.

Destacando apenas alguns artigos de trabalhos publicados [15] [16] [17] [18] [19], de entre os inúmeros que se podem encontrar que tirariam partido de uma biblioteca destes módulos reutilizáveis para interface com o utilizador, verifica-se assim o potencial de aplicação deste trabalho de dissertação no meio académico e não só.

3.2 Comparação com outros trabalhos de natureza similar

Foram poucos os trabalhos encontrados que sejam directamente comparáveis com o trabalho desenvolvido para esta dissertação, como já referido acima encontram-se esses núcleos IP para venda ou como tarefas a desenvolver por alunos em cursos universitários, outros trabalhos são demasiado rígidos uma vez que foram desenvolvidos para uma aplicação específica. Segue-se uma breve comparação com outros trabalhos encontrados, centrando-se o interesse nos módulos em comum com os criados na Universidade de Aveiro. Estes serão descritos em pormenor nas secções posteriores (3.3), podendo verificar-se que as limi-

tações encontradas nos outros trabalhos analisados foram largamente suplantadas pela flexibilidade e facilidade de uso proporcionada pelos módulos desenvolvidos.

3.2.1 The University of Queensland

Na Universidade de Queensland, Austrália, é possível encontrar uma página *online* de projectos, *"VHDL XSV Board Interface Projects"* [20].

O módulo controlador PS/2 é bidireccional funcionando com um relógio de 1 MHz, deste modo os dados recebidos ou enviados são passados à cadência de 1 *byte* por segundo o que é manifestamente lento quando se recebem/enviam sequências de maior dimensão, esta limitação pode ser contornada com algumas alterações mas o código está espalhado por 14 ficheiros tornando-o mais complicado de perceber e alterar.

O controlador VGA implementa apenas a parte de sincronização dos sinais que vão para o monitor e a leitura de uma RAMDAC (RAM + *Digital-to-Analog Converter*) para a geração de cores. Não tem suporte para geração de caracteres de texto ou gráficos e a resolução de imagem máxima é 800x600 @ 72 Hz com um relógio a 50 MHz, podem ser usadas outras resoluções com alteração de parâmetros mas a fonte de valores indicada para estas alterações não é a VESA (*Video Electronics Standards Association*), é indicado também que a imagem criada no monitor não é nítida.

3.2.2 OpenCores.org

O OpenCores.org [21] é um repositório *online* de projectos de código aberto para *hardware* reconfigurável e sistemas embutidos mantido por uma comunidade de contribuidores que tem como objectivo o *design* e publicação de núcleos IP gratuitos.

É possível encontrar um controlador PS/2 e um controlador VGA, ambos em linguagem Verilog, o controlador VGA é bastante versátil e já apresenta alguma maturidade uma vez que vai na versão v2.0, permanece no entanto a especificação dos parâmetros de resolução de imagem que tem que ser introduzidos pelo utilizador.

3.2.3 ALSE – Advanced Logic Synthesis for Electronics

ALSE [22] é uma empresa francesa que faz desenvolvimento de produto e consultadoria na área de projectos em FPGA, CPLD e ASIC. Oferece uma pequena gama de núcleos IP gratuitos onde se encontra um controlador PS/2 apenas unidireccional e um filtro de contactos (*debouncer*) que tem a desvantagem de ter que ser ligado a um relógio de 20 Hz em vez do relógio de sistema (tipicamente MHz).

3.2.4 Georgia Institute of Technology

Nos Estados Unidos da América, em Atlanta, encontra-se o Georgia Tech, numa página *Web* do *Cooperative Analog/Digital Signal Processing Laboratory* [23] estão disponíveis vários módulos em VHDL comuns aos deste trabalho de dissertação.

Pode aceder-se a um filtro de contactos que também é limitado pelo relógio que deve ser fornecido, 100 *Hz*, a modificação é possível mas o método que é usado implicaria a utilização de inúmeros registos usando frequências mais altas.

O controlador de teclado PS/2 na realidade não é um controlador de teclado, é um controlador PS/2 simples que só recebe dados série e entrega na saída 1 *byte*, portanto não é possível enviar comandos ao teclado e os códigos correspondentes a cada tecla não são devidamente mapeados, seria necessário implementar essa funcionalidade num componente adicional.

O controlador VGA apenas efectua a sincronização dos sinais RGB (*Red + Green + Blue*) com os sinais de sincronismo que vão para o monitor, não há *buffer* de memória de imagem, nem suporte de texto ou gráficos, está implementado apenas para um relógio de 25 *MHz* e a mudança de frequência requer alterações significativas no código, como é típico de muitas outras implementações, a única resolução de imagem permitida é 640×480 @ 60 *Hz*, novamente são possíveis outros modos de imagem se o utilizador tiver disponíveis os dados para o correcto sincronismo e mediante alteração dentro do código fonte VHDL.

O módulo que implementa o rato PS/2 sofre de duas das limitações comuns ao controlador VGA, as suas implementações são rígidas e só funcionam com um relógio de 25 *MHz* e numa resolução de 640×480 @ 60 *Hz*, o posicionamento do ponteiro está confinado a 640 colunas por 480 linhas de *pixels*.

3.2.5 Digilent Inc.

Esta conhecida marca comercializa serviços de *design* de *hardware* e *software* para o mercado educacional e industrial, oferece um leque de plataformas com FPGAs de gamas mais baixas até às gamas de alta performance, com a vantagem acrescida de um preço com desconto académico, pelo que uma das suas placas, o modelo *NEXYS 2* [24], se tornou bastante popular e alguns alunos adquirido placas próprias, sendo usadas nas disciplinas de Sistemas Digitais Reconfiguráveis, Linguagens de Descrição de *Hardware* e ainda Computação Reconfigurável, leccionadas no Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e que também possui várias destas placas de entre outras.

Está disponível no *web site* da Digilent [25] um projecto para *download* que é usado na verificação e teste das funcionalidades dos periféricos desta placa, nele pode encontrar-se um bloco de nível esquemático (VgaPs2Demo.sch) para o programa ISE (*Integrated Software Environment*) da Xilinx, neste bloco encontram-se um controlador PS/2, um controlador para o rato e um controlador VGA, todos eles muito bem implementados e documentados, novamente as carências persistentes são a rigidez da implementação uma vez que foram especificamente desenhados e optimizados para esta placa, a mudança de relógio implicaria uma alteração profunda em todos os códigos, mais uma vez a única resolução gráfica disponibilizada é 640×480 @ 60 *Hz*.

3.3 Módulos reutilizáveis desenvolvidos

O desenvolvimento dos módulos apresentados foi precedido da adequada investigação dos protocolos e características, passando por algumas diferentes versões evolutivas em performance e funcionalidade, procurando encontrar também um balanço equilibrado entre a flexibilidade através de parametrização, o particionamento da funcionalidade dos blocos em dimensões adequadas e a facilidade de leitura/compreensão do código fonte.

3.3.1 Filtro de contactos

Os botões de pressão e interruptores existentes nas PCBs (e outros contactos mecânicos), quando pressionados ou comutados, não produzem imediatamente uma ligação perfeita, os contactos batem e ressaltam algumas vezes causando oscilações no sinal antes de repousar no seu estado *on* ou *off*, a este efeito chama-se *contact bouncing*, torna-se portanto necessário filtrar estas oscilações (efectuar *debouncing*) e tornar o sinal bem definido no domínio digital.

Esta filtragem pode ser feita com um filtro RC passa-baixo de modo a suavizar o sinal, ou um Shmitt-trigger, a filtragem pode também ser feita por *software* quando usado um microcontrolador ou processador, através da leitura periódica do sinal, este é considerado filtrado após se manter no mesmo estado algumas leituras sucessivas [26]. Este último método pode ser utilizado na lógica reprogramável da FPGA quando a placa não tem filtragem nos interruptores e botões de pressão.

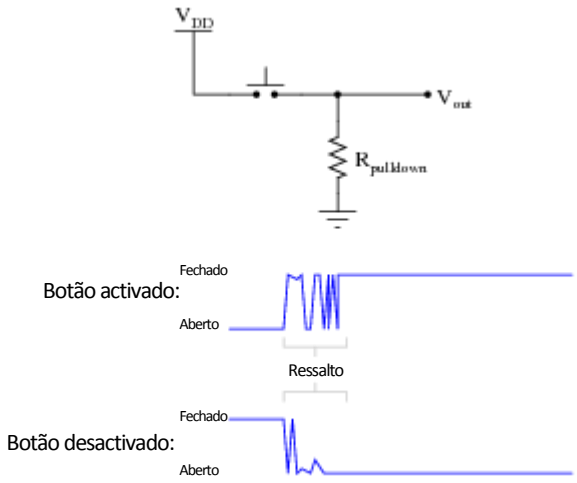


Figura 3.1 – Sinal oscilante num contacto mecânico.

3.3.1.1 O módulo **Debouncer.vhd**

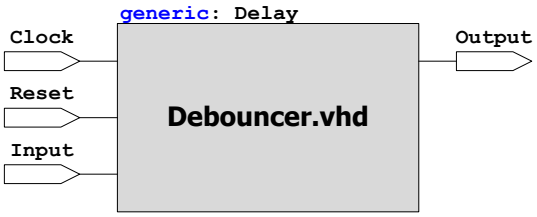


Figura 3.2 – Diagrama do módulo Debouncer.vhd.

Este módulo tem um parâmetro configurável pelo utilizador na entidade VHDL através da palavra reservada `generic`, esse parâmetro especifica o número de ciclos de relógio que o sinal de entrada se deve manter estável para que possa ser passado à saída, é portanto um contador de ciclos de relógio que reinicia quando a entrada muda, se a entrada não mudar o contador incrementa até ao valor do parâmetro e a saída é considerada estável. Existe também um efeito secundário benéfico, o tempo entre escritas no porto de saída permite resolver qualquer eventual estado de meta-estabilidade, embora a probabilidade de permanência no estado meta-estável seja sempre diferente de zero, esta diminui exponencialmente ao longo do tempo [27] [28].

Inevitavelmente é causado um atraso correspondente ao tempo de oscilação mais o tempo de estabilização, no entanto, dada a natureza dos eventos assíncronos como carregar um botão, este atraso não é significativo, deve no entanto ser mantido pequeno quando possível para que não pareça que o sistema não está a responder ao utilizador (e não usar registos enormes), dependendo do tipo e qualidade dos botões, o tempo de oscilação é normalmente curto na ordem dos poucos *ms* ou menos, um tempo de resposta de 100 *ms* aparenta ser imediato ao utilizador, um tempo de estabilização de entre 5 a 10 *ms* é adequando na maioria dos casos.

Por exemplo, a uma frequência de relógio de 50 *MHz*, 5 *ms* correspondem a uma contagem de 250 mil ciclos de relógio, o registo contador teria 18 *bits*.

A interface é muito simples, no porto `Clock` liga-se o relógio de sistema, no porto `Reset` liga-se o sinal de *reset* activo alto, o sinal a filtrar no porto `Input` e o sinal filtrado sai no porto `Output`.

Este módulo pode também ser usado para filtrar outros sinais vindos do exterior da placa, mesmo que a natureza da fonte não seja mecânica, neste caso o tempo de estabilização pode ser bastante reduzido e ainda assim eliminar ruído no sinal e falsas transições, isto é bastante útil no caso de ligações com fios/cabos não blindados que apanham interferências electromagnéticas.

3.3.2 Controlador PS/2

A interface de conexão PS/2 foi introduzida em 1987 pela IBM na sua linha de computadores pessoais *Personal System/2* ou PS/2, o mesmo conector passou a ser usado para o teclado e o rato substituindo os antigos e tem sido usado desde então em computadores compatíveis com a arquitectura IBM PC, contudo caíram em desuso com o aparecimento destes periféricos com conectores USB mas ainda se encontram estas interfaces em PCs e outros equipamentos (bastantes placas de teste com FPGA).

Os conectores são do tipo Mini-DIN (*Deutsches Institut für Normung e.V.* – Instituto Alemão de Estandardização) de 6 pinos mostrados na Figura 3.3, os sinais correspondentes a cada pino estão enumerados na Tabela 3.1.

Figura 3.3 – Conectores PS/2, macho e fêmea.

Tabela 3.1 – Descrição dos sinais PS/2.

Pino	Nome	Descrição
1	DATA	Sinal de dados
2	n/c	Não conectado
3	GND	Massa
4	V_{CC}	Alimentação, +5V
5	CLK	Sinal de relógio
6	n/c	Não conectado

As linhas de dados e de relógio são de colector-aberto com resistências *pull-up* a V_{CC}, isto significa que os dispositivos só tem dois estados, nível lógico '0' ou alta impedância 'Z', quando em alta impedância a linha é elevada a V_{CC} pela resistência *pull-up*, quando inactiva a linha está ao nível lógico '1'.

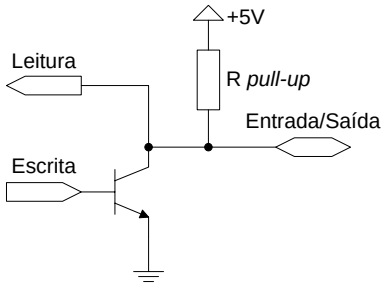


Figura 3.4 – Circuito em colector-aberto.

O protocolo de linha para a comunicação entre interfaces PS/2 é do tipo série, síncrono e bidireccional, todos os dados são transmitidos 1 *byte* de cada vez numa trama de 11 ou 12 *bits*, conforme a sequência na Tabela 3.2.

O hospedeiro tem sempre prioridade sobre o comando das linhas de dados e de relógio, o periférico apenas pode iniciar uma transferência no estado *Idle* – inactividade, o hospedeiro pode retirar o controlo ao periférico em qualquer instante activando a '0' a linha de relógio, estado *Inhibit Communication* – impedir comunicação. O periférico gera sempre o sinal de relógio, com uma frequência de 10 a 16.7 *kHz*. Para estabelecer uma comunicação no sentido hospedeiro → periférico, o hospedeiro tem que colocar as linhas num estado que o periférico reconheça para produzir o relógio e receber dados, o estado *Host Request-to-Send* – hospedeiro requer envio.

Tabela 3.2 – Função dos *bits* na trama PS/2.

Bit #	Função
1	*bit* início (sempre '0')
2	*bit* 0 de dados (menos significativo)
3	*bit* 1 de dados
4	*bit* 2 de dados
5	*bit* 3 de dados
6	*bit* 4 de dados
7	*bit* 5 de dados
8	*bit* 6 de dados
9	*bit* 7 de dados (mais significativo)
10	*bit* de paridade (paridade ímpar)
11	*bit* de paragem (sempre '1')
12	*bit* de acuso de recepção (sempre'0') (*a*)

(*a*) – apenas na comunicação hospedeiro → periférico

Tabela 3.3 – Sumário dos estados da ligação PS/2.

Estado	Data	Clock
Idle	'1'	'1'
Inhibit Communication	'X'	'0'
Host Request-to-Send	'0'	'1'

3.3.2.1 Comunicação periférico → hospedeiro

Neste modo o periférico controla ambas as linhas, na linha de dados os *bits* são escritos quando o relógio está a '1' e lidos pelo hospedeiro no flanco descendente do relógio, os *bits* são enviados em sequência, do 1º ao 11º conforme a Tabela 3.2, e ilustrado na Figura 3.5.

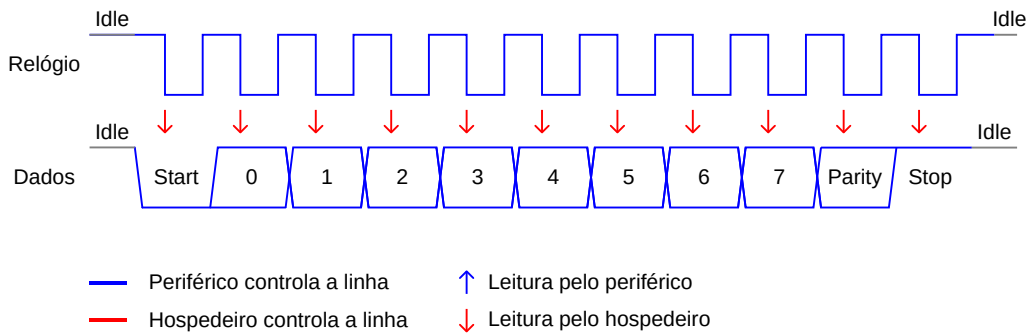


Figura 3.5 – Comunicação periférico → hospedeiro.

3.3.2.2 Comunicação hospedeiro → periférico

Neste modo o hospedeiro leva a linha de relógio a '0' forçando o estado *Inhibit Communication* durante 100 μs, em seguida leva a linha de dados a '0' escrevendo o *bit* de início e passando ao estado *Host Request-to-Send*, seguidamente liberta a linha de relógio, nesta altura o periférico deverá começar a gerar o sinal de relógio o que não deverá acontecer mais tarde do que 15 *ms* após o início do estado *Inhibit Communication*, o hospedeiro detecta os flancos descendentes do relógio e escreve sucessivamente os *bits* de dados, de paridade e de paragem, libertando logo de seguida a linha de dados, o periférico ao ler o *bit* de paragem no flanco ascendente do relógio, assume o controlo da linha de dados e escreve o *bit* de acuso de recepção que o hospedeiro vai ler imediatamente a seguir no flanco descendente. O processo é facilmente compreensível através da Figura 3.6.

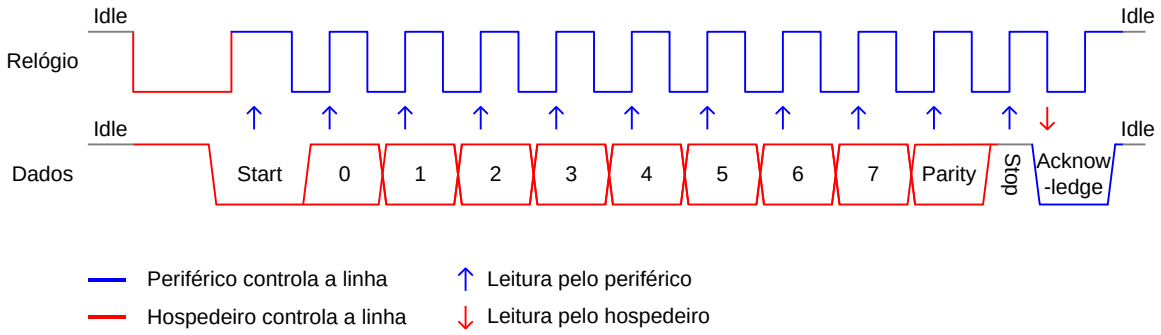


Figura 3.6 – Comunicação hospedeiro → periférico.

Os factos expostos sobre a interface PS/2 são os necessários para a compreensão do ponto de vista do dispositivo hospedeiro, uma vez que o módulo desenvolvido implementa um controlador PS/2 do lado hospedeiro. Particularidades referentes à implementação do lado do periférico tais como dependências temporais entre os sinais de dados e de relógio, bem como os elementos apresentados e outros podem ser encontrados em [29] e [30].

3.3.2.3 O módulo **PS2Controller.vhd**

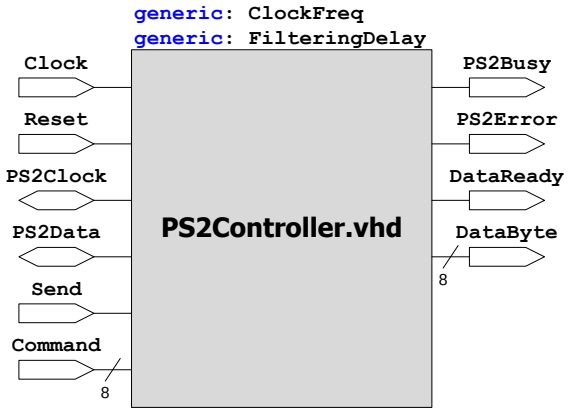


Figura 3.7 – Diagrama do módulo PS2Controller.vhd.

Novamente é utilizada a especificação de parâmetros configuráveis pelo utilizador através da construção VHDL `generic`, é recorrente o uso a esta técnica nos módulos desenvolvidos pois permite flexibilidade na utilização dos módulos em variadas circunstâncias, permite ao utilizador facilmente fazer alterações apenas nos níveis superiores da hierarquia de dependências dos módulos em vez de fazer várias alterações directamente no código e em vários ficheiros separados.

O parâmetro `ClockFreq` especifica em *MHz* a frequência do sinal de relógio que entra no porto `Clock`, é necessário para o cálculo do tamanho dos contadores que implementam a temporização de eventos, por exemplo os 100 μs necessários no estado *Inhibit Communication* na comunicação hospedeiro → periférico.

Um sinal assíncrono de *reset* activo alto liga no porto `Reset`.

Os portos `PS2Clock` e `PS2Data` são bidireccionais e ligam respectivamente à linha de relógio do periférico e linha de dados.

O controlador PS/2 instancia internamente dois módulos Debouncer.vhd para filtrar os sinais `PS2Clock` e `PS2Data` no percurso de entrada, o parâmetro `FilteringDelay` representa o número de ciclos de relógio que a filtragem deve demorar, experimentalmente a 50 *MHz* revelou-se adequado para todos os periféricos PS/2 testados um valor de 15 para o parâmetro `FilteringDelay`, o que corresponde a 320 *ns*, para outras frequências deve ser extrapolado um valor através da equação (1.1) e arredondado ao inteiro mais próximo.

$$Contador\ \ ciclos\ de\ relógio\ = Tempo\ \ \mu s\ \times Frequência\ \ MHz\ \ -1\ \ ciclo\ de\ relógio \quad (1.1)$$

Para o percurso de saída dos sinais `PS2Data` e `PS2Clock` existem *buffers tri-state* (nos IOBs) que conduzem as linhas a alta impedância substituindo assim os transístores em configuração de colector-aberto, não há necessidade de configurar também a resistência *pull-up* nos IOBs da FPGA pois já existe uma do lado do periférico.

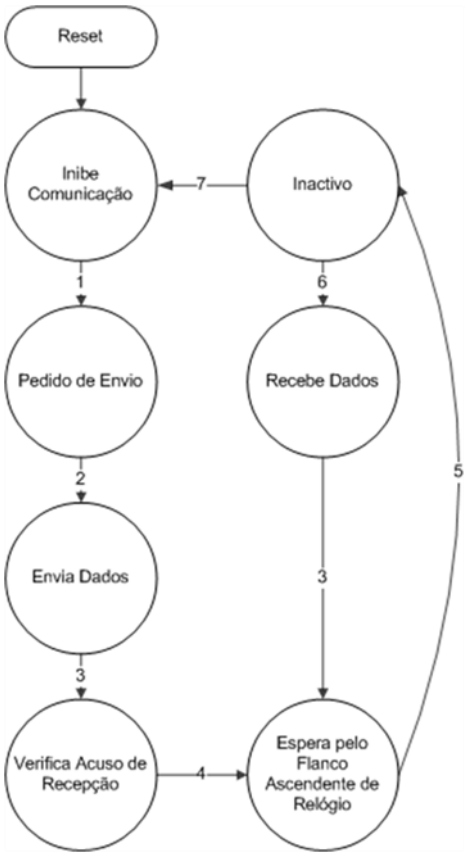
No decorrer duma comunicação de recepção ou envio, o sinal `PS2Busy` encontra-se activo alto, no caso de ser detectado um erro de paridade ou não se detecte o bit de acuso de recepção, o sinal `PS2Error` é activado a '1' e só volta a '0' quando o controlador retornar ao estado de inactividade.

No fim da recepção de um *byte*, este é colocado no porto `DataByte` e o porto `DataReady` é activado a '1' durante um ciclo de relógio.

Para o envio de um comando ao periférico é necessário colocar o *byte* correspondente a esse comando no porto `Command` e activar a '1' o sinal no porto `Send` durante um ciclo de relógio.

A parte de controlo da comunicação é feita através de uma máquina de estados finitos – FSM (*Finite State Machine*), as transições de estados ocorrem sincronamente com o relógio de sistema, no entanto algumas transições de estados são despoletadas pela detecção de um flanco ascendente ou descendente no sinal `PS2Clock`, dependem portanto dos valores na entrada da FSM, como num processo VHDL apenas se pode detectar um flanco (configurado para o relógio de sistema), os flancos do sinal `PS2Clock` são

detectados comparando o valor lógico do sinal no período corrente com o valor do período anterior, se forem diferentes ocorreu um flanco, 0 → 1 flanco ascendente ou 1 → 0 flanco descendente.



1 – Tempo = 100 μs
2 – Tempo = 20 μs
3 – 10º flanco descendente (Stop *bit*)
4 – Flanco descendente
5 – Flanco ascendente
6 – Linha de `PS2Data` = 0
7 – Linha de `Send` = 1

Figura 3.8 – FSM do controlador PS/2.

Existem três tipos de FSM: *Medvedev*, *Moore* e *Mealy*.

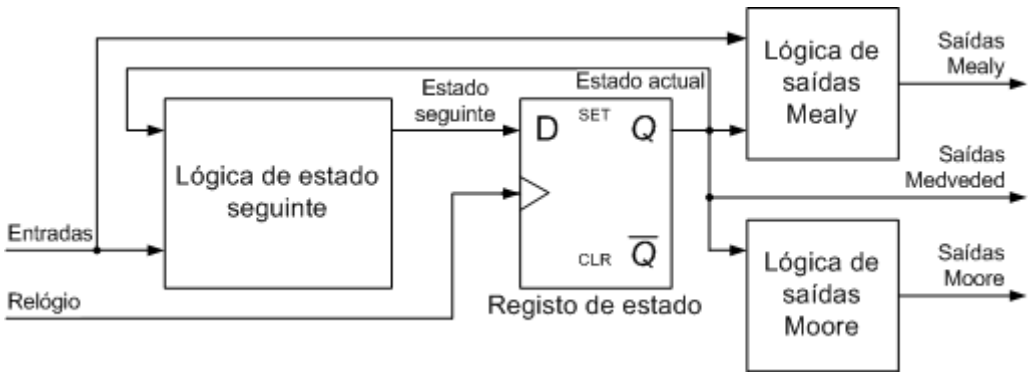


Figura 3.9 – Tipos de FSM.

Nas FSM de Medvedev as saídas coincidem com os valores lógicos dos registos de estado, este tipo de implementação é muito pouco flexível.

Nas FSM de Moore as saídas dependem apenas do estado actual, a sua operação é considerada segura pois os registos mantêm estáveis os sinais das saídas durante pelo menos um ciclo de relógio.

As FSM de Mealy são as mais flexíveis e permitem implementações com menos estados que as anteriores, isto porque as saídas dependem do estado actual e das entradas, existem contudo cuidados a ter, no caso de FSMs encadeadas, as saídas combinatórias de uma FSM de Mealy podem estar ligadas às entradas de outras FSMs de Mealy, como o caminho crítico (e a frequência máxima de operação) depende do atraso causado pela lógica combinatória entre dois flip-flops, a performance pode sofrer uma degradação, outra situação desastrosa acontece quando as saídas de uma FSM de Mealy ligam às entradas de outra e as saídas da segunda ligam às entradas da primeira efectuado uma malha fechada de lógica combinatória, podem suceder duas situações problemáticas, os sinais entram em oscilação descontrolada ou acontece um efeito de memória não desejado.

Uma FSM é tipicamente codificada em VHDL com dois processos, um para a lógica combinatória de determinação do estado seguinte e saídas, e outro processo para a lógica sequencial que guarda e mantém o estado actual. No entanto, como os módulos devem ser reutilizáveis em vários projectos, houve o cuidado de antecipar a ligação com outras FSM, para evitar as malhas fechadas combinatórias, a FSM do controlador PS/2 foi implementada apenas num só processo, deste modo todas as atribuições de sinais são efectuadas no fim do ciclo de relógio pois são inferidos registos para cada sinal, este método provoca um atraso de um período entre a entrada num qualquer estado e a actualização das saídas, tal não é problemático se cada estado durar mais que um ciclo de relógio, o que é o caso. Uma alternativa seria implementar lógica *look-ahead* para determinar as saídas com base nos possíveis estados imediatamente seguintes ao estado actual evitando o atraso de um período, apesar de se tornar difícil implementar este método para FSMs de grandes dimensões ou que não tenham muitas sequências de estados com uma só possibilidade para o estado seguinte.

3.3.3 Teclado PS/2

Os teclados usados hoje em dia têm base no sistema IBM AT (*Advanced Technologies*) de 1984 [31] e no seu sucessor IBM PS/2 de 1987 [32], embora mais avançados ainda suportam as funcionalidades introduzidas nessa altura por razões de retro-compatibilidade. Isto não implica que os teclados actuais obedeçam a estes protocolos como *standard* mas sim por compatibilidade, implementado apenas as características consideradas indispensáveis.

Tabela 3.4 – Teclados IBM e compatíveis [33].

Teclado	Nº de teclas	Conector	Nº de comandos
IBM AT	84-101	DIN 5 pinos	8
IBM PS/2	84-101	Mini-DIN 6 pinos	17
Compatível	Qualquer, normalmente 101-104	Mini-DIN 6 pinos	Pode não implementar todos

As funcionalidades do teclado são implementadas num microcontrolador local, este monitoriza a matriz de teclas verificando quais são pressionadas e/ou libertadas, o microcontrolador é responsável também

por efectuar o *debouncing* dos sinais das teclas, controlar as linhas de dados e de relógio que ligam ao hospedeiro obedecendo ao protocolo de comunicação definido na secção 3.3.2, enviar os códigos respectivos de cada tecla e responder aos comandos enviados pelo hospedeiro.

Durante a inicialização do teclado, tanto por ligação ao hospedeiro ou por reinicialização através de um comando, o teclado executa um teste de funcionamento BAT (*Basic Assurance Test*) durante o qual os três indicadores LED (*Light Emmiting Diode*) estão iluminados, no fim do teste os indicadores são desligados, são carregados os valores por omissão das configurações e é enviada uma mensagem de estado ao hospedeiro, em seguida o teclado inicia o seu normal funcionamento [33].

Existem três conjuntos de códigos para as teclas, denominados *Scan-Code Set 1*, *Scan-Code Set 2* e *Scan-Code Set 3*. O primeiro era adoptado nos sistemas IBM PC e IBM XT (*Extended Technology*) e já não é usado sendo considerado obsoleto. O segundo foi adoptado nos sistemas IBM AT e não suportavam o *Scan-Code Set* 1. O terceiro foi introduzido nos sistemas IBM PS/2 que também suportavam o *Scan-Code Set* 2 mas não o *Scan-Code Set* 1. Nos teclados actuais compatíveis apenas é considerado suportado o *Scan-Code Set* 2 pelo que nem o *Scan-Code Set* 1 nem o *Set* 3 são garantidos e não devem ser usados [34].

No *Scan-Code Set* 2, cada tecla tem um código correspondente quando é pressionada – *make code* – e outro quando é libertada – *break code* – exceptuando a tecla <Pause> que só tem um código *make*.

Para além da emissão de um código quando a tecla é carregada ou libertada existe uma outra situação em que a tecla é mantida pressionada, é enviado o código inicial que corresponde ao pressionamento da tecla, após um pequeno tempo de espera – *typematic delay* – o código é repetido a uma determinada frequência – *typematic rate*, ambos os parâmetros podem ser configurados dentro de certos valores pré-determinados.

Nas 84 teclas originais os *make codes* são constituídos por 1 *byte* único a cada tecla, os *break codes* são constituídos por 2 *bytes*: 1 *byte* de prefixo que é sempre igual (F0h) e de seguida a repetição do *byte make code* da tecla.

As teclas adicionais para além das 84 são consideradas uma extensão ao teclado original, neste caso os seus *make codes* e *break codes* são precedidos também por um prefixo constante (E0h) e denominados por *extended make codes* de 2 *bytes* e *extended break codes* de 3 *bytes* respectivamente.

Os códigos correspondem às posições das teclas no teclado e não aos símbolos, letras e números nelas impressas, deste modo a mesma arquitectura do teclado pode ser usada em inúmeros países com diferentes características de escrita, a correspondência entre o carácter e o código da tecla é estabelecido no hospedeiro e não no teclado.

A título de exemplo encontram-se alguns códigos de teclas na Tabela 3.5, no apêndice A pode ser consultada a tabela integral dos códigos do *Scan-Code Set* 2 e um diagrama do teclado Português.

Tabela 3.5 – Exemplos de códigos de teclas no *Scan-Code Set* 2.

		Make code	*Break code*
Original *	Tecla 110	76h	F0 76h
	Tecla 43	5Ah	F0 5Ah
	Tecla 51	31h	F0 31h
Extended	Tecla 108	E0 5Ah	E0 F0 5Ah
	Tecla 83	E0 75h	E0 F0 75h
	Tecla 81	E0 69h	E0 F0 69h

* As teclas originais não são sequenciais de 1 a 84.

Para além dos códigos correspondentes às teclas, existem também outras sequências de dados que o teclado pode enviar ao hospedeiro e vice-versa, trata-se de comandos e mensagens de configuração ou de estado.

3.3.3.1 Comandos e mensagens enviados pelo teclado PS/2

- FEh – *Resend*, é enviado quando um *byte* é recebido com erro de paridade ou quando não é um comando reconhecido.
- FCh – *BAT Failure Code*, corresponde à falha do teste BAT.
- FAh – *Acknowledge*, o teclado envia este *byte* em resposta a qualquer comando válido com excepção de *Echo* ou *Resend*.
- EEh – *Echo*, quando recebe um comando *Echo* do hospedeiro, o teclado responde também com o mesmo código.
- AAh – *BAT Completion Code*, é enviado quando o teclado completa o BAT com sucesso, qualquer outro valor é considerado um erro.
- 00h – *Key Detection Error/Buffer Overrun*, na impossibilidade de determinar as teclas pressionadas, o teclado envia este comando, isto pode ocorrer devido a uma avaria ou quando o *buffer* FIFO de 16 *bytes* do teclado enche.

3.3.3.2 Comandos e mensagens enviados pelo hospedeiro

- FFh – *Reset*, é enviado para reinicializar o teclado, este ao receber o comando responde com FAh e executa o BAT, o teclado de seguida envia o resultado do teste, AAh para sucesso ou FCh para falha, carrega ainda os valores por omissão:
  - *Typematic delay* = 500 *ms*
  - *Typematic rate* = 10,9 *cps* (caracteres por segundo)
  - *Scan Code Set* 2
  - Todas as teclas activadas no modo *Typematic*/*Make*/*Break*
- FEh – *Resend*, pedido de reenvio, o teclado responde reenviando o último *byte* enviado anteriormente, a excepção é o caso do último *byte* enviado pelo teclado também ter sido FEh, neste caso o teclado envia o último *byte* anterior antes de ter enviado FEh. Este comando é usado para o hospedeiro indicar ao teclado que houve um erro na recepção.

- FDh – *Set Key Type Make*, apenas válido no *Scan Code Set* 3, desliga o envio de código *break* e repetição *typematic* de uma tecla, após receber este comando o teclado responde com FAh, de seguida o hospedeiro deve enviar um código *make* correspondente à tecla a alterar, finda a recepção do código *make* o teclado responde novamente com FAh.
- FCh – *Set Key Type Make/Break*, idêntico ao comando anterior mas apenas desliga a repetição *typematic*.
- FBh – *Set Key Type Typematic*, idêntico ao comando anterior mas apenas desliga o envio do código *break*.
- FAh – *Set All Keys Typematic/Make/Break*, apenas válido no *Scan Code Set* 3, o teclado responde com FAh e configura todas as teclas no seu modo de funcionamento normal com emissão de códigos *make*, *break* e repetição *typematic*.
- F9h – *Set All Keys Make*, apenas válido no *Scan Code Set* 3, o teclado responde com FAh e desliga a emissão de códigos *break* e repetição *typematic* para todas as teclas.
- F8h – *Set All Keys Make/Break*, idêntico ao comando anterior mas apenas desliga a repetição *typematic* para todas as teclas.
- F7h – *Set All Keys Typematic*, idêntico ao comando anterior mas apenas desliga o envio de códigos *break* para todas as teclas.
- F6h – *Set Default*, na recepção deste comando o teclado responde com FAh e carrega a configuração por omissão:
    - *Typematic delay* = 500 *ms*
    - *Typematic rate* = 10,9 *cps*
    - *Scan Code Set* 2
    - Todas as teclas activadas no modo *Typematic*/*Make*/*Break*
- F5h – *Disable*, este comando faz com que o teclado responda com FAh e pare de verificar o estado das teclas e envie os códigos correspondentes, carrega a configuração por omissão e aguarda por novos comandos.
- F4h – *Enable*, o teclado responde com FAh e retoma o normal funcionamento depois de anteriormente ter recebido o comando *disable*.
- F3h – *Set Typematic Rate/Delay*, o teclado responde com FAh e aguarda 1 *byte* de configuração para os parâmetros *typematic*, na recepção do *byte* de configuração o teclado também responde com FAh. Os valores de configuração possíveis encontram-se nas duas tabelas seguintes. O *bit* 7 é sempre 0b.

Tabela 3.6 – Tempos de espera até iniciar a repetição *typematic*.

Bits 6-5	*Delay* (*ms*)
00b	250
01b	500
10b	750
11b	1000

Tabela 3.7 – Valores de repetição *typematic*.

Bits 4-0	*Rate* (*cps*)	*Bits* 4-0	*Rate* (*cps*)	*Bits* 4-0	*Rate* (*cps*)	*Bits* 4-0	*Rate* (*cps*)
00000b	30,0	01000b	15,0	10000b	7,5	11000b	3,7
00001b	26,7	01001b	13,3	10001b	6,7	11001b	3,3
00010b	24,0	01010b	12,0	10010b	6,0	11010b	3,0
00011b	21,8	01011b	10,9	10011b	5,5	11011b	2,7
00100b	20,7	01100b	10,0	10100b	5,0	11100b	2,5
00101b	18,5	01101b	9,2	10101b	4,6	11101b	2,3
00110b	17,1	01110b	8,6	10110b	4,3	11110b	2,1
00111b	16,0	01111b	8,0	10111b	4,0	11111b	2,0

- F2h – *Read ID*, este comando pede um código de identificação ao teclado, este responde com FAh e em seguida envia 2 *bytes* de identificação, primeiro ABh e depois 83h.
- F0h – *Set Scan Code Set*, configura o conjunto de códigos *scan* a utilizar, o teclado responde com FAh e aguarda 1 *byte* de configuração ao qual o teclado responde novamente com FAh, os códigos de configuração possíveis são: 01h que indica a utilização do *Scan Code Set* 1, 02h para o *Scan Code Set* 2 e 03h para o *Scan Code Set* 3. Se o código enviado for 00h o teclado responde com 1 *byte* que indica qual o *Scan Code Set* que está a utilizar no momento: 01h, 02h e 03h para os *Scan Code Set* 1, 2 e 3 respectivamente.
- EEh – *Echo*, teste de conectividade, o teclado responde enviando também *echo*, EEh.
- EDh – *Set/Reset LEDs*, estabelece o estado dos indicadores LED do teclado, este responde com FAh e aguarda 1 *byte* de configuração ao qual responde também com FAh. O *byte* tem o seguinte formato:

Tabela 3.8 – Configuração dos indicadores LED.

Bits 7-3	*Bit* 2	*Bit* 1	*Bit* 0
00000b	*Caps Lock*	*Num Lock*	*Scroll Lock*
	0b - desactivado 1b - activado		

3.3.3.3 O módulo KeyboardMapper.vhd

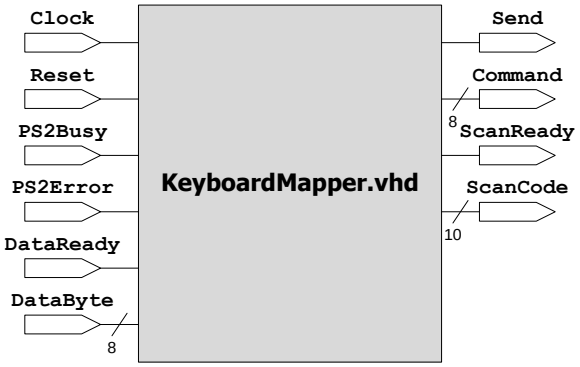


Figura 3.10 – Diagrama do módulo KeyboarMapper.vhd.

Este módulo efectua o mapeamento dos *bytes* de dados recebidos do controlador PS/2 em códigos *scan* das teclas correspondentes, ordena a reinicialização do teclado quando o sistema é ligado, na reinicialização do sistema e ainda quando é detectado um erro no controlador PS/2 ou quando o teste BAT falha. Para além do comando de reinicialização, envia também o comando de configuração dos indicadores LED do teclado e o respectivo *byte* de argumento quando as teclas <Caps Lock>, <Num Lock> ou <Scroll Lock> são pressionadas. Detecta e responde também ao comando de reenvio emitido pelo teclado.

O conjunto de códigos *scan* suportado é o *Scan Code Set* 2 com códigos *make*/*break* e repetição *typematic*. Suporta todas as teclas com excepção das teclas <Pause> e <Print Screen> pois estas tem códigos *make*/*break* que não correspondem ao formato das restantes uma vez que excedem 3 *bytes* de tamanho.

O módulo KeyboardMapper.vhd não tem dependências de outros ficheiros mas foi desenvolvido para ser utilizado em conjunto com o módulo PS2Controller.vhd, no entanto pode ser usado com outro controlador PS/2 desde que respeitadas as características das interfaces.

O sinal de relógio que liga ao porto `Clock` deve ser o mesmo que liga ao controlador PS/2.

Um sinal assíncrono de *reset* activo alto liga no porto `Reset`.

No porto `PS2Busy` deve ser ligado um sinal que deve estar activo alto durante todo o tempo que o controlador PS/2 estiver a transmitir ou a receber dados.

O porto `PS2Error` é assíncrono e nele deve ser ligado um sinal activo alto proveniente do controlador PS/2 que sinalize qualquer erro detectado, não tem duração mínima ou máxima, quando este sinal retoma ao nível lógico '0' é enviado um comando de reinicialização ao teclado.

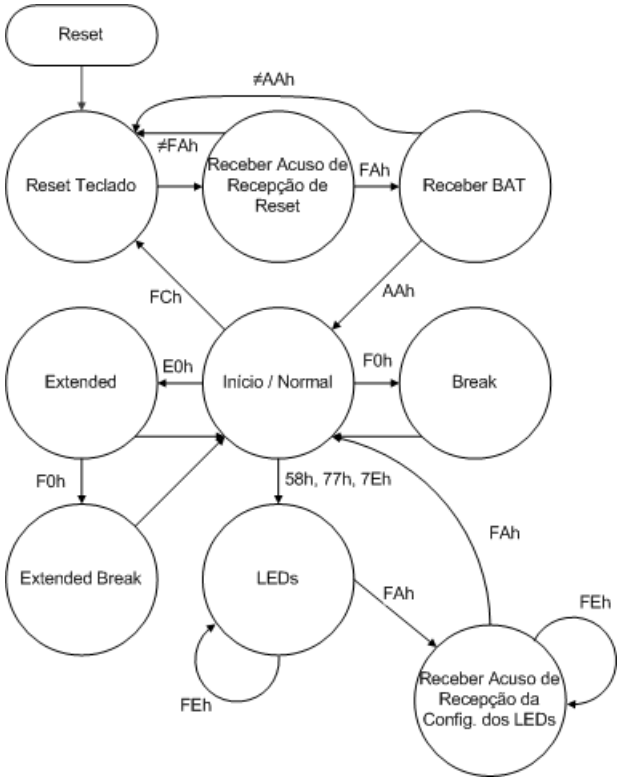
Quando o controlador PS/2 tiver recebido um *byte*, deve coloca-lo no porto `DataByte` e activar o sinal no porto `DataReady` durante apenas 1 ciclo de relógio.

No envio de um comando ao teclado é colocado o *byte* correspondente a esse comando no porto `Command` e activado a '1' o sinal no porto `Send` durante 1 ciclo de relógio.

Quando recebidos todos os *bytes* que compõem um código *scan*, este é colocado no porto `ScanCode` e é activado a '1' o porto `ScanReady` durante 1 ciclo de relógio. Em vez de disponibilizar o código *scan* nos seus formatos de 1, 2 ou 3 *bytes* conforme se trate de códigos *make* ou *break*, normais ou extendidos, o que ocuparia um porto de 24 *bits*, optou-se por efectuar uma optimização para 10 *bits* uma vez que os prefixos são sempre os mesmos para os códigos *break*, F0h, ou extendidos, E0h. Deste modo os códigos *scan* obedecem à seguinte forma:

Tabela 3.9 – Formato dos códigos *scan* do módulo KeyboardMapper.vhd.

	Bit 9	*Bit* 8	*Bits* 7-0
Make	0b	0b	restante código da tecla
Break	0b	1b	
Extended make	1b	0b	
Extended break	1b	1b	



AAh – Teste BAT bem sucedido
FAh – Acknowledge
FCh – Erro no teclado
E0h – Extended
F0h – Break
FEh – Reenviar
58h – Num Lock
77h – Caps Lock
7Eh – Scroll Lock

Figura 3.11 – FSM do controlador de teclado.

3.3.3.4 O módulo **Keyboard.vhd**

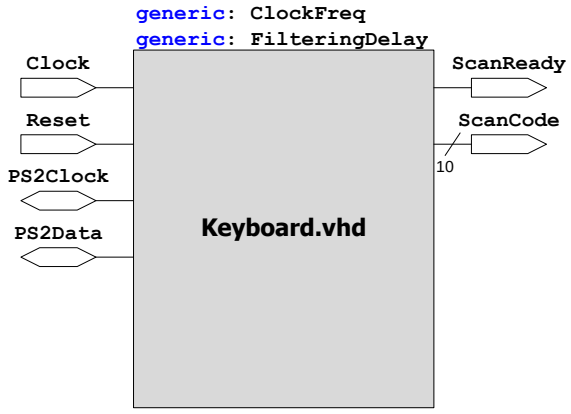


Figura 3.12 – Diagrama do módulo Keyboard.vhd.

Para maior simplicidade e facilidade de utilização, foi criado um módulo de hierarquia superior que instancia os módulos PS2Controller.vhd e KeyboardMapper.vhd, naturalmente as definições dos portos e parâmetros `generic` correspondem às mesmas já indicadas nas secções anteriores.

3.3.3.5 O módulo **KeyboardText.vhd**

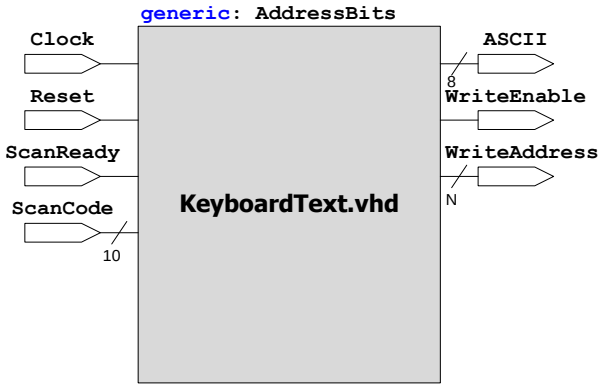


Figura 3.13 – Diagrama do módulo KeyboardText.vhd.

Este módulo transforma códigos *scan* em códigos ASCII (*American Standard Code for Information Interchange*).

É mantido um estado interno das teclas <Num Lock>, <Caps Lock>, <Shift> e <Alt Gr>, deste modo são enviados os códigos ASCII que correspondem às combinações de teclas com os seus símbolos e letras ou números associados assemelhando-se ao funcionamento de um terminal de texto.

Para todas as teclas ou combinações de teclas que correspondam a um código ASCII é enviado o respectivo código para o porto de saída `ASCII` e o porto `WriteEnable` é activado a '1' durante 1 ciclo de relógio.

Sem usar o porto `WriteAddress`, o módulo pode ser usado para passar códigos ASCII a outros blocos para manipulação, visionamento em *displays* de 7 segmentos, LEDs ou *displays* LCD (*Liquid Crystal Display*), etc. A disponibilidade do porto `WriteAddress` tem sentido para a escrita em memória RAM embutida, essa memória é usada pelo controlador VGA como memória *framebuffer* de imagem/texto, isto tornar-se-á claro quando for abordado o monitor VGA na secção seguinte. O porto `WriteAddress` é de largura variável, N *bits* que são especificados no parâmetro genérico `AddressBits` que deve ser o número de *bits* de endereçamento da memória RAM embutida que vai ser usada.

Novamente, um sinal assíncrono de *reset* activo alto liga no porto `Reset` e o sinal de `Clock` deve ser o mesmo usado no módulo Keyboard.vhd.

3.3.4 Controlador de monitor VGA

O modo VGA também foi introduzido pela IBM na sua linha de computadores PS/2 em 1987, sendo uma evolução dos modos CGA (*Colour Graphics Adapter*) e EGA (*Enhanced Graphics Adapter*) também introduzidos anteriormente pela IBM.

Atribuindo um peso relativo às três cores primárias aditivas, RGB (*Red*, *Green*, *Blue*), vermelho, verde e azul, são constituídas as diferentes cores que são mostradas no ecrã do monitor.

O modo CGA tem 4 *bits*, um para cada uma das três cores e um *bit* de intensidade, permite um total de 2^4=16 cores mas apenas 4 ao mesmo tempo em modo gráfico.

O modo EGA tem 6 *bits*, um para cada uma das três cores e um para a intensidade de cada cor, permite um total de 2^6=64 cores, das quais 16 podem ser usadas simultaneamente em modo gráfico.

Os modos CGA e EGA são completamente digitais com níveis TTL (*Transistor-Transistor Logic*) de 5 V, o monitor encarrega-se de converter a informação digital em sinais analógicos que controlam os canhões de electrões que estimulam a superfície fosforescente do ecrã.

O modo VGA envia ao monitor a informação de cor de forma analógica, 0 V corresponde à cor preto e 0,7 V à cor branco, ou seja todas as três cores primárias na sua intensidade máxima, através do sinal analógico o número de cores possíveis de mostrar torna-se ilimitado devido à continuidade do sinal. As cores continuam a ser codificadas em formato digital e posteriormente convertidas numa DAC no controlador VGA e só depois enviadas ao monitor. O standard VGA usa 6 *bits* para cada uma das três cores primárias totalizando 18 *bits* de codificação de cor, são possíveis então 2^{18}=262.144 cores das quais podem ser usadas apenas 256 simultaneamente no modo gráfico de menor resolução (320×200) ou somente 16 no modo gráfico de maior resolução permitido pelo standard, 640×480 *pixels* [35].

O controlador VGA era embarcado na placa principal do computador, no entanto o standard permitia uma extensão através de um adaptador vídeo auxiliar mas que acedia aos recursos existentes do subsistema vídeo de base [35]. A IBM logo lançou uma placa com acelerador gráfico, a IBM 8514 executava algumas funções gráficas libertando o CPU desse trabalho e aumentava a resolução para 1024×768 *pixels* com 256 cores simultâneas. Rapidamente fabricantes competidores começaram a introduzir as suas próprias versões modificadas e melhoradas com desvios ao standard da IBM.

Devido à explosão de modos gráficos não estandardizados, em 1989 nasceu a VESA (*Video Electronics Standards Association*) [36], uma associação que tem como objectivo a criação de standards para a indústria dos monitores e adaptadores gráficos.

Como os modos gráficos sucessores tomaram como base o funcionamento do mesmo subsistema vídeo, VGA passou igualmente a designar não só o modo gráfico introduzido pela IBM mas também qualquer standard de vídeo analógico para PC que use a mesma técnica, a diferenciação atinge-se com base na resolução gráfica utilizada, VGA-640×480, SVGA-800×600, XGA-1024×768, etc...

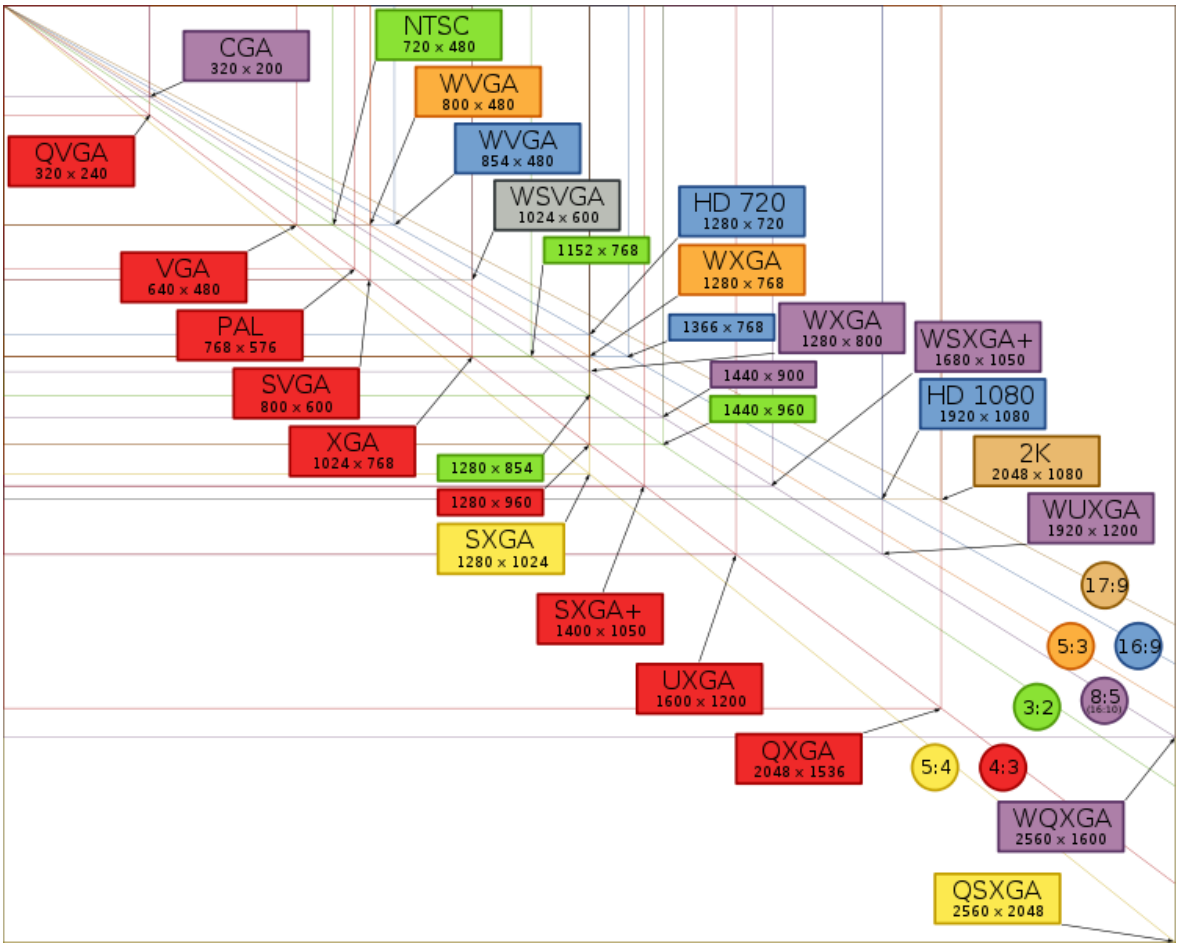


Figura 3.14 – Várias resoluções gráficas.

3.3.4.1 Interface física

Os primeiros controladores e monitores VGA herdaram os conectores D-Subminiature DE-9 de 9 pinos também usados nos controladores e monitores CGA e EGA. Com os 9 pinos era suficiente para acomodar os sinais necessários.

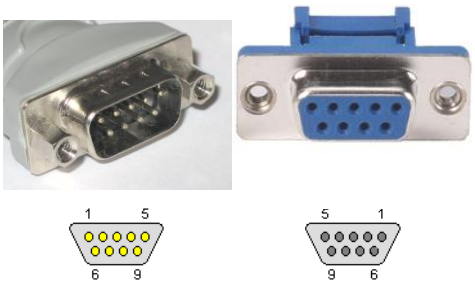


Figura 3.15 – Conectores DE-9 macho e fêmea.

Tabela 3.10 – Descrição dos sinais VGA no conector DE-9.

Pino	Nome	Direcção	Descrição
1	RED	→	Vídeo, vermelho
2	GREEN	→	Vídeo, verde
3	BLUE	→	Vídeo, azul
4	HSYNC	→	Sincronismo horizontal
5	VSYNC	→	Sincronismo vertical
6	RGND		Massa, vermelho
7	GGND		Massa, verde
8	BGND		Massa, azul
9	SGND		Massa, sincronismo

Na Tabela 3.10, Tabela 3.11 e Tabela 3.12, a direcção especificada é sempre no sentido do controlador VGA para o monitor.

Em seguida passou a ser usado o conector DE-15 compacto, os pinos adicionais permitiam identificar se o monitor ligado ao controlador era a cores ou monocromático e ainda se suportava resoluções iguais ou superiores a 1024×768.

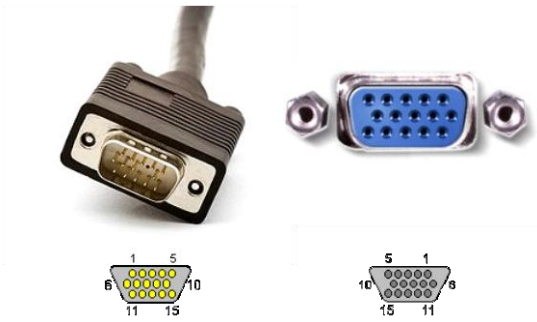


Figura 3.16 – Conectores DE-15 macho e fêmea.

Tabela 3.11 – Descrição dos sinais VGA no conector DE-15 compacto.

Pino	Nome	Direcção	Descrição
1	RED	→	Vídeo, vermelho
2	GREEN	→	Vídeo, verde
3	BLUE	→	Vídeo, azul
4	ID2	←	ID do monitor, *bit* 2
5	GND		Massa
6	RGND		Massa, vermelho
7	GGND		Massa, verde
8	BGND		Massa, azul
9	KEY		Sem pino, sem ligação
10	SGND		Massa, sincronismo
11	ID0	←	ID do monitor, *bit* 0
12	ID1	←	ID do monitor, *bit* 1
13	HSYNC	→	Sincronismo horizontal
14	VSYNC	→	Sincronismo vertical
15	ID3	←	ID do monitor, *bit* 3

Mais tarde a VESA introduziu o DDC (*Display Data Channel*) que permite comunicação bidireccional entre o monitor e o controlador, assim é possível o monitor identificar-se perante o controlador e ainda o controlador pode ajustar parâmetros do monitor, tais como brilho, contraste, tonalidade e ajustes de margens, etc., bem como ajustar o modo de funcionamento, normal, *standby* ou *power saving*, através de alguns comandos enviados num protocolo análogo ao I^2C (*Inter-Integrated Circuit*).

Tabela 3.12 - Descrição dos sinais VGA no conector DE-15 compacto, versão DDC2.

Pino	Nome	Direcção	Descrição
1	RED	→	Vídeo, vermelho
2	GREEN	→	Vídeo, verde
3	BLUE	→	Vídeo, azul
4	RES		Reservado
5	GND		Massa
6	RGND		Massa, vermelho
7	GGND		Massa, verde
8	BGND		Massa, azul
9	+5V	→	$+5V_{DC}$
10	SGND		Massa, sincronismo
11	ID0	←	ID do monitor, opcional
12	DAS	↔	DDC, linha dados série
13	HSYNC	→	Sincronismo horizontal
14	VSYNC	→	Sincronismo vertical
15	SCL	↔	DDC, linha relógio série

3.3.4.2 Geração de imagens

Um *pixel* consiste na menor amostra de informação (cor ou tom) de uma imagem, o termo *pixel* deriva da contracção de duas palavras, *pix-pictures* e *el-elements*. Um *pixel* é sempre uma tríade de elementos de cor RGB, dependendo da tecnologia do monitor os *pixels* podem ter diferentes arranjos como se verifica na imagem seguinte.

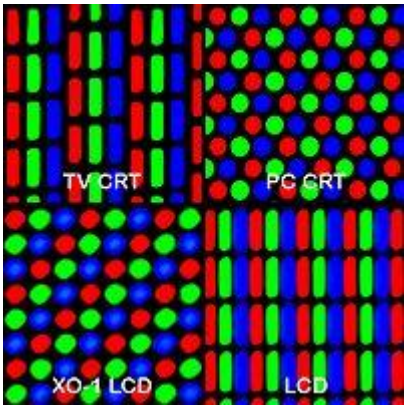


Figura 3.17 – Diferentes arranjos de elementos RGB.

Os monitores apresentam a imagem em duas dimensões, portanto uma imagem é reconstruída a partir de uma matriz de x×y *pixels*. A reconstrução é efectuada pela ordem normal de índices de uma matriz rectangular, numa linha são varridos os elementos da esquerda para a direita e para todas as linhas de cima para baixo. Este processo é repetido várias vezes por segundo, denomina-se taxa de refrescamento de imagem e tem valores típicos que rondam os 50 e os 120 *Hz*.

Como um *pixel* apenas comporta a informação de cor e não a sua localização na imagem, o controlador do lado do monitor tem que ter um mecanismo suplementar que permita activar o *pixel* correcto no ecrã.

Isto é conseguido com dois sinais adicionais para sincronismo, um horizontal e um vertical, e ainda um tempo preciso para a activação de cada *pixel*.

Deste modo, a informação de cada linha de *pixels* é enviada ao monitor obedecendo à seguinte forma:

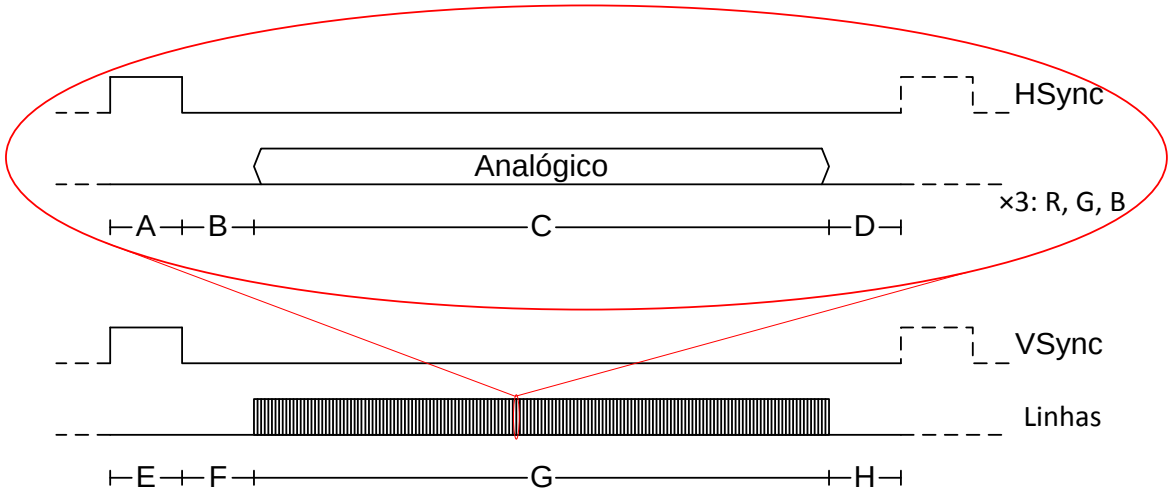


Figura 3.18 – Sincronização dos sinais VGA.

	Designação
A	Pulso de sincronização horizontal
B	Banda de guarda esquerda
C	Linha de imagem
D	Banda de guarda direita
E	Pulso de sincronização vertical
F	Banda de guarda de topo
G	Janela de imagem
H	Banda de guarda de fundo

Cada linha começa com um pulso de sincronização horizontal (A) seguido de uma banda de guarda (B) do lado esquerdo – *Horizontal Back Porch*, em seguida são activados os três sinais analógicos RGB que indicam a coloração de cada *pixel*, todos os *pixels* de uma linha são coloridos no período C, finalmente a linha termina numa banda de guarda do lado direito (D) – *Horizontal Front Porch*. Os tempos A+B+D servem para o ajuste do monitor para a linha seguinte. Este procedimento repete-se durante todo o período de uma janela de imagem, E+F+G+H, no entanto, apenas durante G os sinais RGB serão diferentes de 0 *V*. Portanto o tempo total de uma janela de imagem não é dado apenas pelo tempo da janela visível. Como se pode ver na Figura 3.18 e Figura 3.19, também existe um sinal de sincronismo vertical COM e duas bandas de guarda, no topo – *Vertical Back Porch*, e de fundo – *Vertical Front*

Porch. Ambos os sinais de sincronismo, vertical e horizontal podem ter polaridades positivas ou negativas dependendo do modo a visualizar (Tabela 3.13 – Modos gráficos do standard VGA.).

Através da GTF (*Generalized Timing Formula*) [37] da VESA podem ser calculados os valores de A, B, C, D, E, F, G e H quando especificada a resolução do ecrã e uma das seguintes frequências: de pixel, de linha ou de janela.

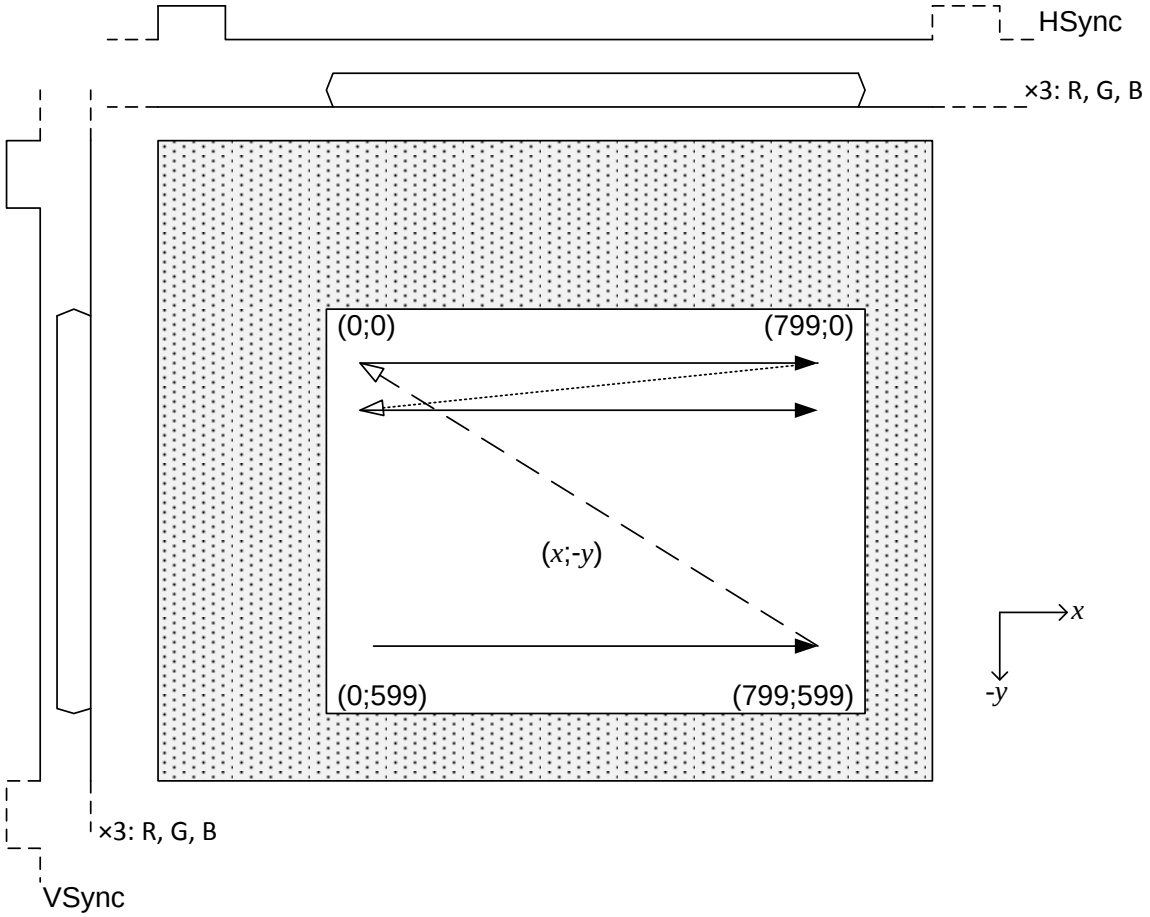


Figura 3.19 – Janela de imagem, inclui janela visível e as bandas de guarda.

O sistema base VGA da IBM compreendia apenas 256 kB de memória vídeo, distribuídos em 4 blocos de 64 kB cada, a informação de cor dos *pixels* era armazenada nessa memória, em modo gráfico a organização da memória e a forma como a informação é formatada dependem do modo utilizado. Os modos e algumas das suas características são apresentados na Tabela 3.13.

A cada ciclo de relógio a memória é lida e os 18 *bits* de RGB são enviados para a DAC e convertidos em 3 sinais analógicos, a frequência usada é seleccionada através de dois *bits* num registo particular do sistema de vídeo base. Para os modos gráficos APA (*All Points Addressable*) e alguns modos alfanuméricos é 25,125 MHz, para algumas resoluções de 720×y ou 360×y que existem apenas em modo alfanumérico a frequência usada é de 28,322 MHz.

Em suma, os principais componentes do sistema base de vídeo que controla o monitor são:

- relógio do sistema vídeo;
- gerador de sinais de sincronismo;
- memória *buffer* de vídeo;
- conversor digital-analógico.

Tabela 3.13 – Modos gráficos do standard VGA.

Mode (hex)	Type	Colors	Alpha Format	Buffer Start	Box Size	Max. Pgs.	Freq.	Vert. PELs
0,1	A/N	16	40x25	B8000	8 x 8	8	70Hz	320x200
0*,1*	A/N	16	40x25	B8000	8x14	8	70Hz	320x350
0 +, 1 +	A/N	16	40x25	B8000	9x16	8	70Hz	360x400
2,3	A/N	16	80x25	B8000	8x8	8	70Hz	640x200
2*,3*	A/N	16	80x25	B8000	8x14	8	70Hz	640x350
2 +, 3 +	A/N	16	80x25	B8000	9x16	8	70Hz	720x400
4,5	APA	4	40x25	B8000	8x8	1	70Hz	320x200
6	APA	2	80x25	B8000	8x8	1	70Hz	640x200
7	A/N	–	80x25	B0000	9x14	8	70Hz	720x350
7 +	A/N	–	80x25	B0000	9x16	8	70Hz	720x400
D	APA	16	40x25	A0000	8x8	8	70Hz	320x200
E	APA	16	80x25	A0000	8x8	4	70Hz	640x200
F	APA	–	80x25	A0000	8x14	2	70Hz	640x350
10	APA	16	80x25	A0000	8x14	2	70Hz	640x350
11	APA	2	80x30	A0000	8x16	1	60Hz	640x480
12	APA	16	80x30	A0000	8x16	1	60Hz	640x480
13	APA	256	40x25	A0000	8x8	1	70Hz	320x200

Note: * or + Enhanced modes

3.3.4.3 A *package* Monitor.vhd

Esta *package* foi criada com a ideia de parametrização em mente e de levar mais além as capacidades do controlador VGA standard, disponibilizando modos gráficos de resoluções superiores e maiores taxas de refrescamento.

Nesta *package* estão definidos tipos e estruturas de dados que facilitam a utilização do sistema de vídeo desenvolvido.

Foi definido um tipo composto que define constantes para 18 modos gráficos com várias resoluções e que devem ser escolhidos com base no relógio de sistema ou relógio de vídeo. As constantes definidas resultam da utilização da GTF para calcular os tempos de A, B, C, D, E, F, G e H da Figura 3.18, os valores correspondem ao número de ciclos de relógio necessários para a duração desses tempos, C e G definem a resolução horizontal e vertical respectivamente uma vez que cada ciclo de relógio é a duração de um *pixel*, é definida também a polaridade dos sinais de sincronismo. Assim sendo, não é necessário definir estes parâmetros nem ter acesso à GTF, basta escolher um dos vários modos definidos.

```
	type Direction is (Horiz, Vert);
	type TimingInfo is
		record
			Displayed    : natural;
			BackPorch    : natural;
			FrontPorch   : natural;
			SyncPulse    : natural;
			SyncPolarity : STD_LOGIC;
		end record;
	type Mode is array (0 to 17, Direction'left to Direction'right) of Timin-
gInfo;

	constant Monitor : Mode := (
                                        …
	-- 36MHz Clock          (85Hz)     ----------------
	(( 640, 112,  32,  48, '0'),     -- MONITOR  4 --
	 ( 480,  25,   1,   3, '0')),    ----------------
                                        …
	);
```

Na *package* são incluídos também os mapas de *bits* que constituem os caracteres e são indexados pela sua posição na tabela ASCII, deste modo são facilmente lidos especificando esse índice como endereçamento numa memória inicializada com os mapas de *bits*, tem uma dimensão de altura de 12 *pixels* e largura de 8 *pixels*.

```
	type ROM is array (0 to 126, 0 to 11) of STD_LOGIC_VECTOR(0 to 7);

	constant PixelMap : ROM := (

                                        …
		65 => ("00000000",  --  A
		       "00110000",
		       "01111000",
		       "11001100",
		       "11001100",
		       "11001100",
		       "11111100",
		       "11001100",
		       "11001100",
		       "11001100",
		       "00000000",
		       "00000000"),
                                        …
		others => (others => x"00") );
```

São definidas também outras constantes usadas pelo sistema, como o número total de caracteres possíveis de mostrar no ecrã, caracteres por linha de texto, quantas linhas de texto e ainda as dimensões das células de texto, estas últimas podem ser modificadas mas o utilizador terá de redesenhar novos caracteres (ou outros gráficos) para as novas dimensões.

```
-- Enter your selected monitor mode HERE
--                                ||
--                                \/
constant SELECTED : natural :=       8; -- everything else adjusts accordingly

constant HBP : natural := Monitor(SELECTED, Horiz).BackPorch;        -- Horizontal Back Porch end
constant HAV : natural := HBP + Monitor(SELECTED, Horiz).Displayed;  -- Horizontal Active Video end
constant HFP : natural := HAV + Monitor(SELECTED, Horiz).FrontPorch; -- Horizontal Front Porch end
constant HP  : natural := HFP + Monitor(SELECTED, Horiz).SyncPulse;  -- Horizontal Sync Pulse end
constant HSP : STD_LOGIC := Monitor(SELECTED, Horiz).SyncPolarity;   -- Horizontal Sync Polariry
constant VBP : natural := Monitor(SELECTED, Vert).BackPorch;         -- Vertical Back Porch end
constant VAV : natural := VBP + Monitor(SELECTED, Vert).Displayed;   -- Vertical Active Video end
constant VFP : natural := VAV + Monitor(SELECTED, Vert).FrontPorch;  -- Vertical Front Porch end
constant VP  : natural := VFP + Monitor(SELECTED, Vert).SyncPulse;   -- Vertical Sync Pulse end
constant VSP : STD_LOGIC := Monitor(SELECTED, Vert).SyncPolarity;    -- Vertical Sync Polariry
constant CHARWIDTH   : natural := 8;
constant CHARHEIGHT  : natural := 12;
constant TOTALPIXELS : natural := Monitor(SELECTED, Horiz).Displayed * Monitor(SELECTED,
Vert).Displayed;
constant TEXTROWS    : natural := Monitor(SELECTED, Vert).Displayed / CHARHEIGHT;
constant TEXTCOLUMNS : natural := Monitor(SELECTED, Horiz).Displayed / CHARWIDTH;
constant TOTALCHARS  : natural := TEXTROWS * TEXTCOLUMNS;
```

É definida também uma pequena paleta de 8 cores em formato RGB (enumeradas pelo nome da cor , *white*, *cyan*, *blue*, etc.) para especificar nas constantes apropriadas a cor do texto e cor de fundo, ou para serem usadas no design, por exemplo o cursor do rato.

```
	type RGB is
		record
			R : STD_LOGIC;
			G : STD_LOGIC;
			B : STD_LOGIC;
		end record;
	type BasicColours is (Black, Blue, Green, Cyan, Red, Magenta, Yellow, White);
	type ColourRGB is array (BasicColours'left to BasicColours'right) of RGB;
	constant COLOUR : ColourRGB := (

	('0', '0', '0'),  -- Black
	('0', '0', '1'),  -- Blue
	('0', '1', '0'),  -- Green
	('0', '1', '1'),  -- Cyan
	('1', '0', '0'),  -- Red
	('1', '0', '1'),  -- Magenta
	('1', '1', '0'),  -- Yellow
	('1', '1', '1')); -- White
```

3.3.4.4 O módulo VGASync.vhd

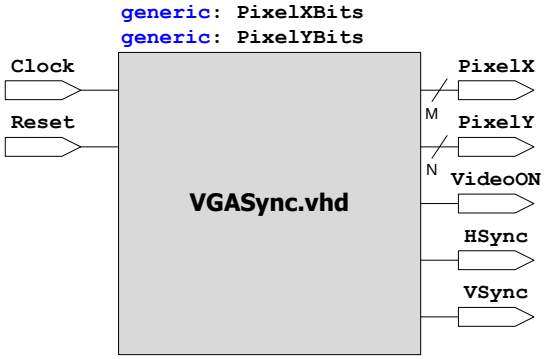


Figura 3.20 – Diagrama do módulo VGASync.vhd

A geração de sinais de sincronismo `HSync` e `VSync` é efectuada com dois contadores, o contador primário incrementa a cada ciclo de relógio e reinicializa ao fim do tempo de uma linha do ecrã, comparadores verificam se o valor se encontra dentro dos valores de contagem definidos para o período do pulso de sincronismo horizontal e activam o porto `HSync` com a polaridade correcta. O processo para o porto `VSync` é análogo com o contador secundário a incrementar a cada linha do ecrã completa.

Os portos `PixelX` e `PixelY` indicam no sistema de coordenadas qual a posição do *pixel* que está a ser varrido no ecrã, quer seja um *pixel* da área visível ou não, servem também para calcular o endereço de memória onde estará a informação RGB do *pixel*. Os parâmetros genéricos `PixelXBits` e `PixelY-Bits` indicam a largura dos barramentos dos portos `PixelX` e `PixelY`, isto porque consoante a resolução usada são necessárias mais ou menos linhas, trata-se de uma pequena medida de optimizar a implementação, estes valores são calculados automaticamente segundo a resolução escolhida.

Se o *pixel* for da área visível, o porto `VideoON` é activado a '1' ou '0' em caso contrário.

No porto `Clock` deve ser ligado o relógio de vídeo e o sinal de `Reset` é assíncrono activo alto.

3.3.4.5 O módulo **RGBMux.vhd**

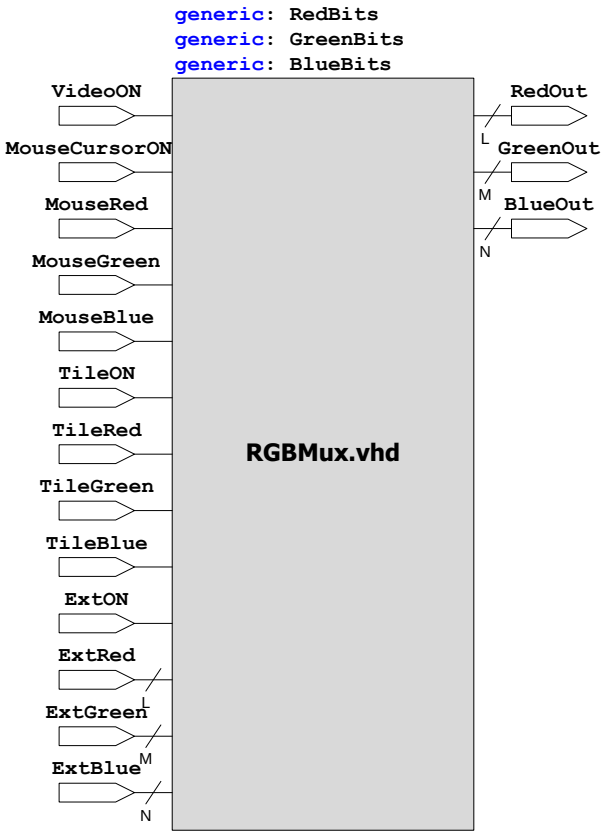


Figura 3.21 – Diagrama do módulo RGBMux.vhd.

Os parâmetros `RedBits`, `GreenBits` e `BlueBits` definem o número de *bits* usados em cada cor e a largura dos portos de entrada e saída referentes a informação de cor, em muitas placas de FPGA existem DACs vídeo com diferentes profundidades de cor, por exemplo em *bits* R-G-B, a Digilent NEXYS2 tem 3-3-2, a Celoxica RC10 tem 7-7-7 enquanto a Xilinx ML507 tem 8-8-8.

O módulo funciona como um misturador da informação de cor de várias fontes, uma para o cursor do rato, uma para as células (*tiles*) de cada carácter de texto ou outros pequenos gráficos mapeados do mesmo modo, e ainda uma outra fonte externa que pode ser por exemplo uma memória com informação RGB de uma imagem.

O porto `VideoOn` do módulo VGASync.vhd liga ao módulo RGBMux.vhd para dar indicação de passagem de informação às saídas `RedOut`, `GreenOut` e `BlueOut`. Cada fonte de informação RGB tem o seu respectivo sinal que indica se está activa ou não, por exemplo `MouseON`, além disso todas as fontes tem prioridades atribuídas para as saídas RGB, o cursor do rato fica sempre em cima de todas as imagens logo tem a maior prioridade, em seguida as *tiles* e por último a fonte externa.

Note-se que poderá haver perda de sincronismo devido aos sinais RGB serem gerados mais tarde que os sinais `HSync` e `VSync`, neste caso os sinais de RGB deverão ser sincronizados novamente com os sinais

`HSync` e `VSync` atrasando estes últimos o número de ciclos de relógio necessários para a geração dos sinais RGB.

3.3.4.6 O módulo SymbolROM.vhd

Figura 3.22 – Diagrama do módulo SymbolROM.vhd.

Esta é a memória onde são carregados os mapas de *bits* dos caracteres definidos na *package*, o porto `SymbolPos` corresponde ao código ASCII do carácter a visualizar, como o ecrã é varrido linha a linha de cima para baixo, é também necessário especificar no porto `LineAddress` qual das 12 linhas da célula vai ser varrida, no porto `ROMData` são colocados os 8 *bits* que pertencem a essa linha.

3.3.4.7 O módulo RAM.vhd

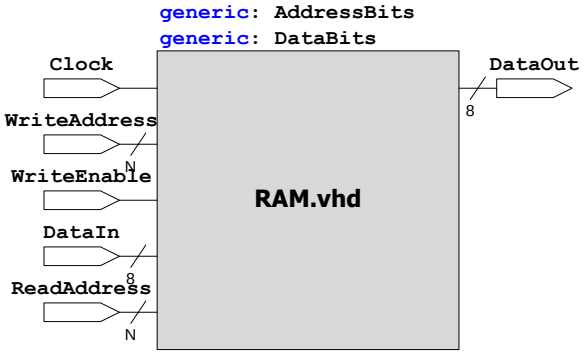


Figura 3.23 – Diagrama do módulo RAM.vhd.

Este módulo implementa em RAM embutida uma memória de tamanho configurável com `AddressBits` do barramento de endereçamento e `DataBits` de largura de dados. No contexto do sistema de vídeo, armazena os códigos ASCII dos caracteres a imprimir no ecrã, portanto a largura de dados é de 8 *bits*, a largura de *bits* do barramento de endereçamento depende do número de caracteres possíveis de colocar no ecrã e que por sua vez depende da resolução escolhida, este cálculo é efectuado na *package* e `AddressBits` é gerado automaticamente.

A memória pode ser inicializada com conteúdo de modo a mostrar texto estático e as posições onde o utilizador pode escrever são endereçadas no porto `WriteAddress` (vindo do módulo Keyboard.vhd) com o conteúdo de `DataIn` (do porto `ASCII` do módulo Keyboard.vhd). O porto `WriteEnable`

permite activar ou desactivar a escrita na memória. `ReadAddress` indica o endereço a ser lido e é calculado no módulo VGATileMatrix.vhd que será abordado de seguida.

O porto `DataOut` envia o conteúdo – que é um código ASCII – ao porto `SymbolPos` do módulo SymbolROM.vhd.

No porto `Clock` deve ser ligado o relógio de vídeo.

3.3.4.8 O módulo VGATileMatrix.vhd

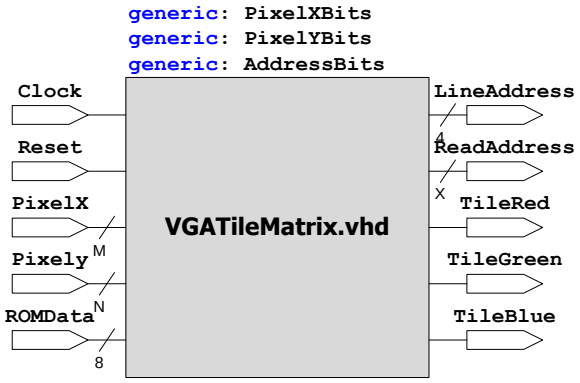


Figura 3.24 – Diagrama do módulo VGATileMatrix.vhd.

Com as posições de `PixelX` e `PixelY` provenientes do módulo VGASync.vhd e conhecendo o tamanho das células de texto (definido na *package*) é determinada a linha da célula que está a ser impressa no ecrã, `ReadAddress` indica ao módulo RAM.vhd a posição do carácter, o código ASCII lido é passado ao módulo SymbolROM.vhd que conjuntamente com `LineAddress` seleccionam os 8 *bits* da linha que é enviada para `ROMData`. Consoante a cor de fundo e de fonte previamente escolhidas, a informação de cor é passada nos portos `TileRed`, `TileGreen` e `TileBlue`, assume-se que 8 cores para texto sejam suficientes de modo que apenas são utilizados 3 *bits* de cor.

O sinal de `Reset` é assíncrono activo alto e o sinal `Clock` deverá ser o relógio de vídeo.

`PixelXBits`, `PixelYBits` e `AddressBits` já foram definidos anteriormente e a sua funcionalidade é idêntica neste módulo e são gerados automaticamente.

3.3.4.9 Geração de dados gráficos

A impressão de figuras no ecrã requer quantidades de memória consideráveis, frequentemente superiores às disponíveis na FPGA quando as figuras tenham uma grande profundidade de cor e/ou sejam de tamanho aproximado a toda a área do ecrã. Torna-se necessário recorrer a memória externa à FPGA, essa memória de *framebuffer* é preenchida com a informação de cor dos *pixels*. Porém a forma de carregar esses dados na memória externa não é trivial se o software de suporte da placa FPGA não oferecer essa possibilidade.

Nas FPGAs baseadas em tecnologia SRAM que são a maioria, há sempre uma memória de suporte onde é armazenado o *bitstream*, se houver ainda espaço nessa memória (tipicamente memória *flash*) os restantes sectores podem ser carregados com as imagens e através de alguma funcionalidade de *bootloader* implementada no design a imagem seria passada da memória *flash* para a memória volátil mais rápida. Poderia também ser usada uma interface de comunicação (ethernet, RS232, etc.) para carregar a memória volátil.

Em computadores digitais modernos o modelo de cor RGB é representado numa base binária, geralmente em 24 *bits* com 8 *bits* por cada cor primária, um *pixel* passa a ocupar 3 *bytes* com informação RGB, embora devido ao formato das arquitecturas seja mais rápido processar blocos alinhados de 32 *bits* o modelo mantém-se, sendo os 8 *bits* adicionais usados para um canal de transparência ou não são usados de todo. Por exemplo, uma imagem de 640×480 *pixels* de 24 *bits* de cor ocuparia: 640×480×3 = 900 *kB* na memória de *framebuffer*.

O formato de imagem digital de computador que mais facilmente se manipula para conseguir extrair a informação RGB de cada *pixel* é o formato BMP ou *Bitmap*, isto porque essa informação é acessível directamente sem descodificação nem descompressão, no entanto a ordenação de informação de cada *pixel* não é da esquerda para a direita e de cima para baixo relativamente ao formato a duas dimensões da imagem, o que requer alguma manipulação.

Num ficheiro BMP com profundidade de cor de 24 *bpp* (*bits* por *pixel*) a informação de cor é armazenada a partir do *offset* 36h, os valores RGB dos *pixels* são organizados como mostra a Tabela 3.14, pode verificar-se que a ordem das cores está invertida, a sequência dos *pixels* ao longo do ficheiro é da esquerda para a direita em linhas de baixo para cima referentes ao plano da imagem.

Tabela 3.14 –Organização de informação de cor num ficheiro BMP.

Offset	+0h	+1h	+2h
36h	B	G	R
39h	B	G	R
3Ch	NULL	NULL	
3Eh	B	G	R
41h	B	G	R
44h	NULL	NULL	
46h	…		

Com esta informação pode ser implementado um *script* simples em qualquer linguagem de programação que ordene os *bytes* em RGB e na sequência em que a imagem é gerada no ecrã, para serem carregados na memória *framebuffer* ou em RAM embutida se houver capacidade suficiente disponível. Neste último caso a inicialização é mais fácil, através do acessório "CORE Generator" do programa ISE da Xilinx.

O *software* de suporte da placa Digilent NEXYS2 tem um aplicativo que permite carregar valores tanto na memória RAM como na memória Flash da placa.

3.3.5 Rato PS/2

Outro dispositivo adaptado ao sistema PS/2 da IBM foi o rato, baseado no modelo anterior com interface série RS-232, o rato manteve o seu mecanismo de detecção de movimento inalterado, o sistema opto-mecânico é baseado numa esfera que roda em contacto com a superfície onde o rato desliza, dois tacómetros ópticos são rodados por discos em contacto com a esfera, o posicionamento perpendicular permite determinar os movimentos em fase e quadratura caracterizando assim o movimento nos sentidos $\pm x$ e $\pm y$.

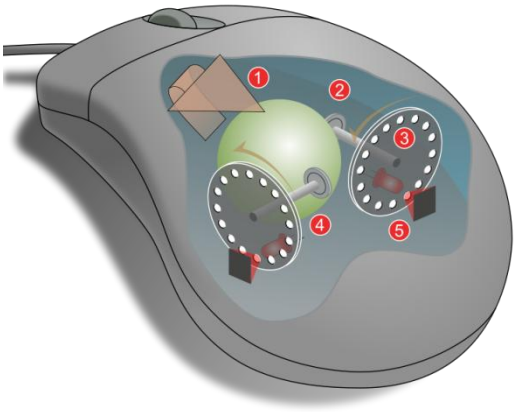


Figura 3.25 – Mecanismo opto-mecânico de detecção de movimento.

Para além dos opto-mecânicos existem também os ratos opto-electrónicos, um LED[†] ilumina a superfície onde o rato desliza, a luz reflectida na superfície é recebida num sensor óptico e um DSP (*Digital Signal Processor*) processa as imagens sequenciais e determina a posição do rato relativamente à posição anterior caracterizando o movimento nos sentidos $\pm x$ e $\pm y$, este tipo de rato tem maior precisão e funciona em quase qualquer superfície, mesmo irregular.

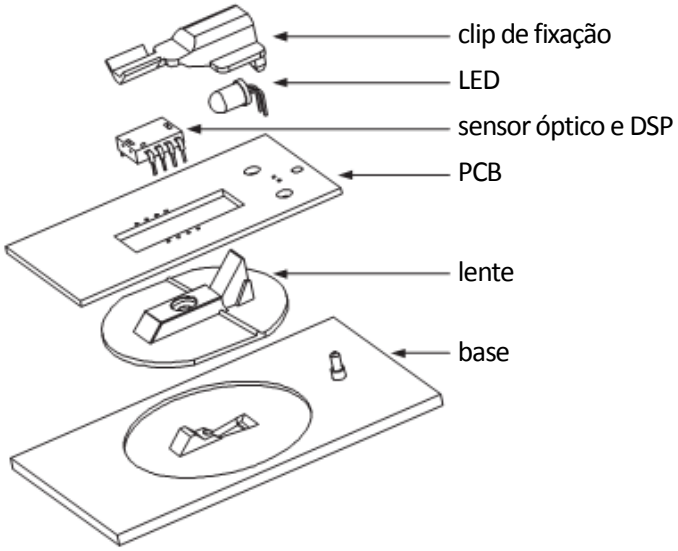


Figura 3.26 – Mecanismo opto-electrónico de detecção de movimento.

[†] Pode ser usado também um díodo de infra-vermelhos ou um díodo laser.

O rato é ligado ao hospedeiro através do mesmo protocolo PS/2 usado pelo teclado, os dados relativos ao movimento são enviados em pacotes de 3 *bytes* na forma mostrada na Tabela 3.15.

Tabela 3.15 – Formato dos pacotes de comunicação de movimento do rato.

	Bit 7	*Bit* 6	*Bit* 5	*Bit* 4	*Bit* 3	*Bit* 2	*Bit* 1	*Bit* 0
Byte 1	*Overflow* Y	*Overflow* X	Sinal Y	Sinal X	'1'	M	R	L
Byte 2	Movimento X							
Byte 3	Movimento Y							

Os pacotes são enviados sempre que é detectado um movimento ou um evento num botão, a cada envio os contadores são reinicializados, o movimento é portanto caracterizado em incrementos ou decrementos relativamente à posição anterior.

Os contadores de movimento discriminam inteiros de 9 *bits* em complemento para 2, os *bits* correspondentes aos sinais de X e Y encontram-se no *byte* 1 do pacote e o restante valor dos contadores é transmitido nos *bytes* 2 e 3, quando o movimento excede a contagem permitida pelos 9 *bits*, de -255 a +254, é sinalizado a '1' o respectivo *bit* de *overflow*, o sentido positivo de X é para a direita e o sentido positivo de Y é em direcção ao utilizador. O estado dos botões é indicado nos *bits* 2 a 0 no primeiro *byte*; M, R e L correspondem respectivamente aos botões do meio, da direita e da esquerda, '1' estado pressionado e '0' estado não pressionado.

Os contadores podem ser incrementados ou decrementados em diferentes resoluções de distância, o valor por omissão é de 4 *contagens/mm* mas pode também ser configurado para 1, 2 ou 8 *contagens/mm*.

É possível também mudar a escala de 1:1 para 2:1, esta modificação ocorre sobre o valor transmitido no pacote de dados e não directamente no valor dos contadores.

Tabela 3.16 – Mudança de escala 2:1.

Contador	Contagem reportada
1:1	2:1
0	0
1	1
2	1
3	3
4	6
5	9
N>5	2×N

O rato pode operar em quatro modos distintos de funcionamento:

Reset – O rato entra neste modo quando é ligado à alimentação ou quando recebe um comando de *Reset* (FFh), o rato executa o teste de diagnóstico BAT e configura-se com os seguintes valores por omissão:

- Ritmo de amostragem = 100 *amostras/s*
- Resolução = 4 *contagens/mm*
- Escala = 1:1
- Reporte de dados desligado

e envia o código do estado do teste BAT, se ocorreu com sucesso (AAh) ou falha (FCh), de seguida envia um código correspondente à identificação (00h) que o distingue de um teclado ou outro tipo de rato em modo estendido[‡]. Após o envio da identificação o rato passa automaticamente ao modo *Stream*, note-se que o reporte de dados é desligado por omissão, requerendo que se envie um comando (F4h) ao rato para que este possa começar a enviar dados de posição.

Stream – Este é o modo normal de funcionamento, se o reporte de dados estiver activado, o rato envia pacotes de 3 *bytes* com informação de movimento e estado dos botões como definido na Tabela 3.15. Os pacotes são enviados apenas quando há alteração de posição ou do estado dos botões, por omissão o ritmo máximo de envio de pacotes é de 100 por segundo, o que ocorre se houver constantemente movimento do rato.

Remote – Neste modo o rato actualiza os contadores e monitoriza o estado dos botões mas não envia os pacotes sempre que há alterações de valores, assim sendo o rato apenas envia um pacote e mediante um pedido do hospedeiro através do comando *Read data* (EBh).

Wrap – Este modo serve para testar a ligação, todos os bytes recebidos pelo rato são reenviados ao hospedeiro, as únicas excepções são os comandos *Reset* (FFh) e *Reset Wrap Mode* (ECh) que retornam o rato ao modo em que se encontrava antes de entrar no modo *Wrap*, um dos modos *Stream* ou *Remote*.

3.3.5.1 Comandos e mensagens enviados pelo rato PS/2

- FEh – *Resend*, é enviado quando um *byte* é recebido com erro de paridade ou quando não é um comando reconhecido.
- FCh – *BAT Failure Code*, corresponde à falha do teste BAT.
- FAh – *Acknowledge*, o rato envia este *byte* em resposta a qualquer comando válido com excepção de *Resend*.
- AAh – *BAT Completion Code*, é enviado quando o rato completa o BAT com sucesso, qualquer outro valor é considerado um erro.
- 00h – *Standard PS/2 Mouse ID*, o rato envia este *byte* em resposta ao comando *Get Device ID* (F2h) ou no final do procedimento de *Reset*.

[‡] Os modos estendidos não fazem parte do protocolo PS/2 original, embora sejam definidos sobre ele, por exemplo o uso da roda ou a possibilidade de usar mais que 3 botões são modos estendidos.

3.3.5.2 Comandos e mensagens enviados pelo hospedeiro

Se o rato estiver no modo *Stream*, o hospedeiro deve desligar o reporte de dados com o comando F5h antes de enviar qualquer outro comando.

- FFh – *Reset*, é enviado para reinicializar o rato, este ao receber o comando responde com FAh e executa o BAT, o rato de seguida envia o resultado do teste, AAh para sucesso ou FCh para falha, carrega ainda os valores por omissão:
    - Ritmo de amostragem = 100 *amostras/s*
    - Resolução = 4 *contagens/mm*
    - Escala = 1:1
    - Reporte de dados desligado
- FEh – *Resend*, o hospedeiro efectua um pedido de reenvio quando recebe um pacote inválido do periférico, o rato responde reenviando o último pacote enviado anteriormente. Este comando é usado para o hospedeiro indicar ao rato que houve um erro na recepção. Se o rato responder com outro pacote inválido a acção seguinte pode ser reenviar novamente o comando *Resend*, enviar um comando de *Reset*, impedir a comunicação ou declarar uma interrupção e servir uma rotina de erro, a acção tomada depende do controlador do hospedeiro.
- F6h – *Set Default*, na recepção deste comando o rato responde com FAh e carrega a configuração por omissão:
    - Ritmo de amostragem = 100 *amostras/s*
    - Resolução = 4 *contagens/mm*
    - Escala = 1:1
    - Reporte de dados desligado

  em seguida reinicializa os contadores de movimento e entra no modo *Stream*.

- F5h – *Disable Data Reporting*, este comando faz com que o rato responda com FAh, desligue o reporte de dados e reinicialize os contadores de movimento. O rato continua a actualizar os contadores embora não envie pacotes, ficando a funcionar em modo semelhante ao modo *Remote*.
- F4h – *Enable Data Reporting*, o rato responde com FAh e reinicializa os contadores de movimento retomando o normal funcionamento no modo *Stream* e enviando pacotes de dados.
- F3h – *Set Sample Rate*, o rato responde com FAh e aguarda 1 *byte* de configuração para ritmo de amostragem, na recepção do *byte* de configuração o rato também responde com FAh. Os valores de configuração possíveis encontram-se na tabela seguinte.

Tabela 3.17 – Valores possíveis para o ritmo de amostragem.

Ritmo de amostragem (*amostras/s*)	*byte* de configuração
10	0Ah
20	14h
40	28h
60	3Ch
80	50h
100	64h
200	C8h

- F2h – *Get Device ID*, o rato responde com FAh e de seguida envia o seu ID, 00h para um rato PS/2 *standard*, deve também reinicializar os contadores de movimento.
- F0h – *Set Remote Mode*, o rato responde com FAh, reinicializa os contadores de movimento e entra no modo *Remote*, enviando pacotes apenas mediante a recepção de um comando explícito do hospedeiro.
- EEh – *Set Wrap Mode*, o rato responde com FAh, reinicializa os contadores de movimento e entra no modo *Wrap*, reenviando ao hospedeiro todos os *bytes* recebidos com excepção dos comandos *Reset* (FFh) e *Reset Wrap Mode* (ECh).
- ECh – *Reset Wrap Mode*, o rato responde com FAh, reinicializa os contadores de movimento e retorna ao modo em que se encontrava antes de entrar no modo *Wrap*, um dos modos *Stream* ou *Remote*.
- EBh – *Read Data*, o rato responde com FAh e envia um pacote de dados reinicializando de seguida os seus contadores de movimento. Este é o único modo do hospedeiro obter pacotes de dados quando o rato se encontra no modo *Remote*.
- EAh – *Set Stream Mode*, o rato responde com FAh, reinicializa os contadores de movimento e entra no modo *Stream*.
- E9h – *Status Request*, o hospedeiro usa este comando para obter o estado em que o rato se encontra a funcionar. Na recepção deste comando o rato envia 1 *byte* FAh e em seguida um pacote de 3 *bytes* com a indicação do seu estado segundo o formato da tabela seguinte.

Tabela 3.18 – Formato dos pacotes de descrição de estado.

	Bit 7	*Bit* 6	*Bit* 5	*Bit* 4	*Bit* 3	*Bit* 2	*Bit* 1	*Bit* 0
Byte 1	'0'	Modo	Reporte	Escala	'0'	M	R	L
Byte 2	Resolução							
Byte 3	Ritmo de amostragem							

Modo = '1' Remote
'0' Stream

Escala = '1' 2:1
'0' 1:1

Reporte = '1' ligado
'0' desligado

Resolução = Valores da Tabela 3.12

Ritmo de amostragem = Valores da Tabela 3.14

M, R, L = '1' botão pressionado
'0' botão não pressionado

- E8h – *Set Resolution*, o rato responde com FAh e aguarda 1 *byte* de configuração para a resolução, na recepção do *byte* de configuração o rato também responde com FAh. Os valores de configuração possíveis encontram-se na tabela seguinte.

Tabela 3.19 – Valores possíveis para a resolução.

byte de configuração	Resolução (*contagens/mm*)
00h	1
01h	2
02h	4
03h	8

- E7h – *Set Scaling 2:1*, o rato responde com FAh e afecta os valores dos contadores de movimento com o factor de escala definido na Tabela 3.16.
- E6h – *Set Scaling 1:1*, o rato responde com FAh e os valores dos contadores de movimento não são afectados por nenhum factor de escala.

3.3.5.3 O módulo **MouseController.vhd**

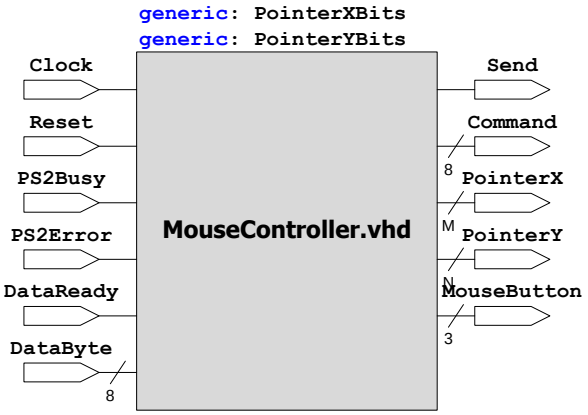


Figura 3.27 – Diagrama do módulo MouseController.vhd.

O controlador do rato liga directamente a um segundo controlador PS/2 idêntico ao do teclado, pelo que os portos `PS2Busy`, `PS2Error`, `DataReady`, `DataByte`, `Send` e `Command` tem as mesmas funcionalidades e já foram cobertos anteriormente.

Os parâmetros genéricos `PointerXBits` e `PointerYBits` definem a largura dos barramentos `PointerX` e `PointerY` que indicam no sistema de coordenadas a localização do ponteiro do rato, assume-se que o cursor é livre de percorrer toda a área visível do ecrã portanto `PointerXBits` e `PointerYBits` são iguais a `PixelXBits` e `PixelYBits` respectivamente, mais uma vez estes parâmetros são gerados automaticamente baseados nas definições de resolução escolhida na *package*.

O porto `MouseButton` tem 3 *bits* de largura, cada *bit* indica o estado de um botão com '0' para não pressionado e '1' para pressionado.

O controlador ordena a reinicialização do rato quando o sistema é ligado, na reinicialização do sistema e ainda quando é detectado um erro no controlador PS/2 ou quando o teste BAT falha. Seguindo ao teste BAT concluído com sucesso é enviado o comando de pedido de identificação do rato que deve responder com 00h, em seguida envia o comando que activa o reporte de dados e após resposta afirmativa do rato, o controlador entra em modo de repetição lendo sempre 3 *bytes* sucessivos e executando os cálculos que determinam a posição do cursor baseada na informação de deslocamento contida nos 3 *bytes* e ainda o estado dos 3 botões.

O sinal `Reset` é assíncrono activo alto e o relógio que deve ser ligado ao porto `Clock` deve ser o mesmo do sistema vídeo.

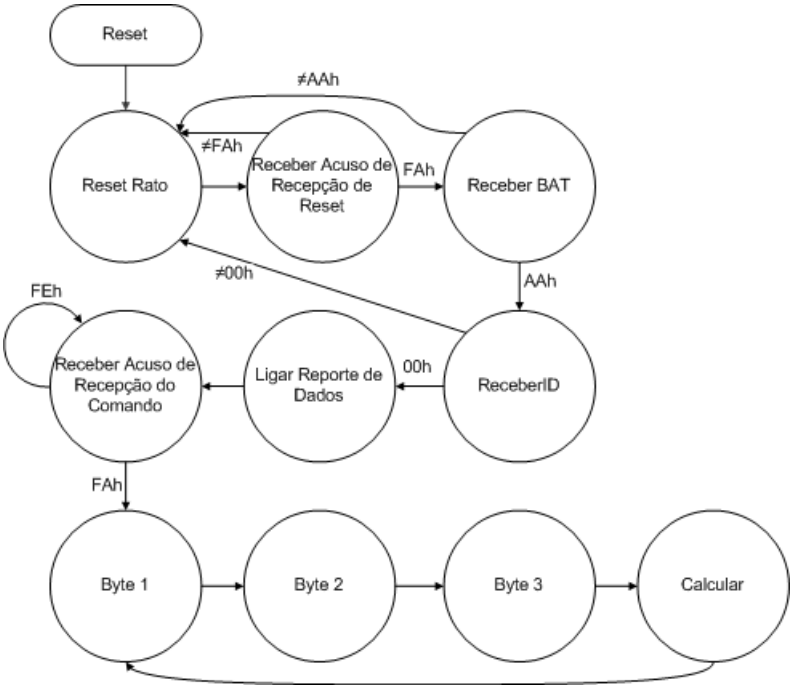


Figura 3.28 – FSM do controlador do rato.

3.3.5.4 O módulo **MouseCursor.vhd**

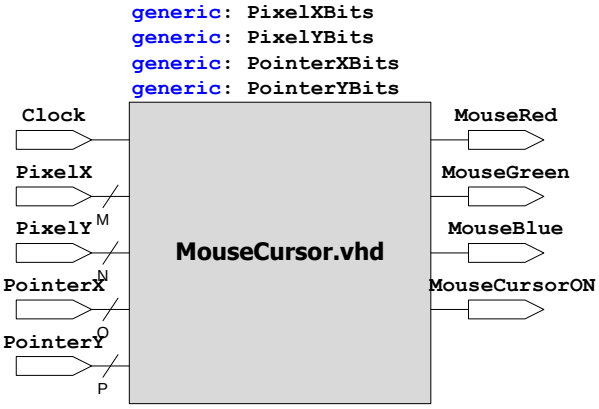


Figura 3.29 – Diagrama do módulo MouseCursor.vhd.

Este módulo gera os sinais RGB para o padrão do cursor do rato, este consiste numa imagem de 16×16 *pixels* incluída numa memória de 256 posições de 2 *bits* cada definida por uma constante com inicialização.

Mediante a comparação do *pixel* activo indicado nos portos `PixelX` e `PixelY` com a posição do ponteiro indicada em `PointerX` e `PointerY`, é determinado se o *pixel* activo se encontra dentro da zona atribuída ao cursor, as saídas `MouseRed`, `MouseGreen` e `MouseBlue` são activadas a branco ou peto conforme o padrão, a transparência é conseguida activando a '0' o porto `MouseCursorON`, caso contrário o ponteiro teria uma forma de 16 por 16 *pixels*.

O relógio a ligar no porto `Clock` deve ser o mesmo relógio do sistema vídeo.

3.3.5.5 O módulo **PS2Mouse.vhd**

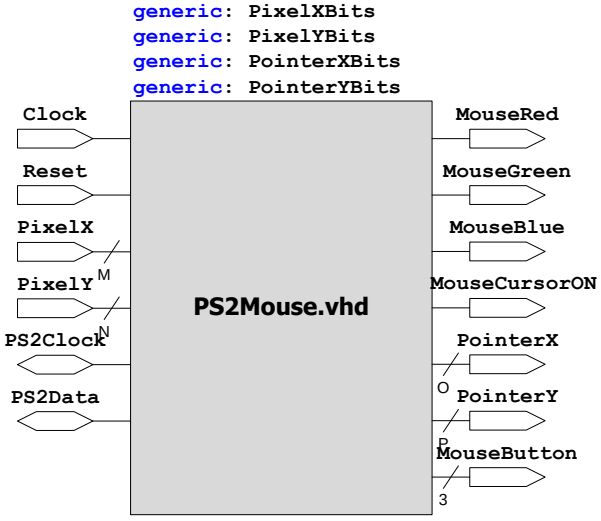


Figura 3.30 – Diagrama do módulo PS2Mouse.vhd.

Assim como efectuado para o teclado, também foi criado um módulo de hierarquia superior que engloba todas as instanciações necessárias à implementação do rato PS/2 para mais fácil utilização.

Este módulo inclui portanto dois filtros de contactos para as linhas PS/2 bidireccionais, um controlador PS/2, o controlador do rato e o gerador de padrão do cursor.

3.3.6 UART

O standard RS-232 [38] dominou a comunicação série entre equipamentos durante décadas, desenvolvido para ligar à distância computadores *mainframe* e tele-máquinas de escrever como dispositivo de saída, através de linhas telefónicas e consequentemente modems. Define-se portanto um DTE (*Data Terminal Equipment*) como o equipamento terminal e um DCE (*Data Circuit-terminating Equipment*) que seria o modem. Para manter a integridade da transmissão sobre a linha telefónica foi desenvolvido um esquema de controlo que requeria 22 sinais ligados através de conectores DB-25.

Na realidade hoje em dia, apenas um subconjunto minimal de sinais do standard é utilizado, especialmente em computadores PC o número de sinais usados na comunicação permitiu a redução dos conectores e linhas nos cabos, inicialmente conduzida pelo aparecimento nos computadores IBM PC/AT como uma modificação não estandardizada, a sua larga proliferação deu origem ao aparecimento do standard TIA-574 [39] que usa conectores DE-9 em vez dos DB-25 definidos no standard original, sendo ainda muitas vezes utilizado para comunicação série entre equipamentos, se bem que já largamente suplantado pelo protocolo USB, continua a ter bastante aplicabilidade em sistemas industriais e sistemas embutidos ou de aplicação específica.

O controlador de comunicação denomina-se UART (*Universal Asynchronous Receiver-Transmitter*) e implementa dois circuitos, um transmissor e um receptor, isto permite comunicação *full-duplex* ou seja transmissão e recepção de dados simultaneamente.

A comunicação é efectuada segundo um esquema assíncrono *start/stop* com um número de *bits* de dados configurável (5, 6, 7 ou 8), número de *bits* de paragem (1, 1,5 ou 2) e opcionalmente um *bit* de paridade par ou ímpar também configuráveis, é usado um relógio padrão (*baud rate*) que também deve ser definido *a priori* assim como os demais parâmetros.

Baud rates típicos: 300, 600, 1200, 1800, 2400, 4800, 7200, 9600, 14400, 19200, 38400, 57600, 115200 Bd.

Cada *bit* tem a duração de 16 ciclos de relógio e o *baud* é gerado através de um divisor de relógio, para os divisores serem números inteiros e não haver erro na geração dos *baud*, a UART usa um oscilador de 22,118400 MHz.

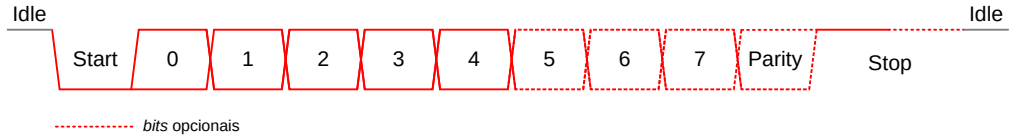


Figura 3.31 – Comunicação assíncrona *start/stop*.

Segundo o standard RS-232 os níveis lógicos estão definidos sobre os níveis de tensão da Tabela 3.20.

Tabela 3.20 – Níveis de tensão para os níveis lógicos em RS-232.

Nível lógico	Níveis de tensão
'1' ou *mark*	-15 a -3 V
indefinido	-3 a +3 V
'0' ou *space*	+3 a +15 V

Os circuitos electrónicos digitais não usam estas gamas de tensão para sinalizar os níveis lógicos pelo que este tipo de comunicação usa *chips* dedicados como a UART 16550 com todos os registos de controlo ou quando o controlo é efectuado num sistema embutido apenas é necessário um conversor de níveis de tensão como o MAX232 da Maxim.

A interface física apresentada corresponde à utilização do conector DE-9 segundo o standard TIA-574 do lado DTE com a direcção dos sinais na Tabela 3.21 indicando o sentido do DTE para o DCE..

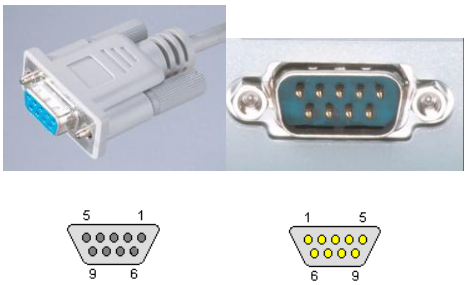


Figura 3.32 – Conectores DE-9 fêmea e macho, no cabo e no DTE.

Tabela 3.21 – Descrição dos sinais no conector DE-9 do lado do DTE.

Pino	Nome	Direcção	Descrição
1	DCD	←	*Data Carrier Detect*
2	RxD	←	*Receive Data*
3	TxD	→	*Transmit Data*
4	DTR	→	*Data Terminal Ready*
5	SGND		*Signal Ground*
6	DSR	←	*Data Set Ready*
7	RTS	→	*Request To Send*
8	CTS	←	*Clear To Send*
9	RI	←	*Ring Indicator*

Os sinais *Receive Data* e *Transmit Data*, como o nome indica, servem para receber e transmitir *bits* em série do DCE segundo o formato da Figura 3.31, quando as linhas estão inactivas encontram-se no estado lógico '1', portanto com tensão negativa.

Request To Send é activado a '0' quando o DTE quer mandar dados ao DCE, quando o DCE estiver livre para receber dados activa o *Clear To Send* a '0' e o DTE pode começar a enviar dados. Estes sinais implementam um esquema de *handshaking* para moderar o fluxo de dados enviados ao DCE.

Data Terminal Ready e *Data Set Ready* são usados para sinalizar que os dispositivos estão conectados e ligados.

Data Carrier Detect é usado para o DCE sinalizar que estabeleceu ligação com o exterior, como os DCE inicialmente eram modems, este sinal servia para indicar a ligação com outro modem do lado oposto da linha telefónica.

Ring Indicator, também relacionado com os modems DCE indica que o modem está a ser contactado pela linha telefónica para estabelecer uma ligação.

É bastante usual encontrar equipamentos que apenas tem implementados os sinais TxD, RxD, RTS e CTS, mesmo só com os sinais RxD e TxD é possível estabelecer comunicação sem usar controlo de fluxo por *handshaking*.

3.3.6.1 O módulo UART.vhd

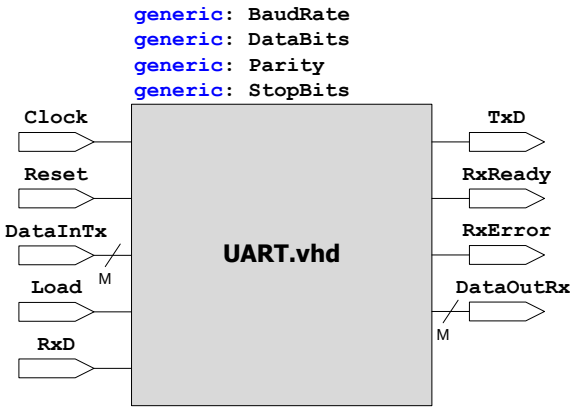


Figura 3.33 – Diagrama do módulo UART.vhd.

Este módulo implementa uma simples UART com capacidade de transmissão e recepção simultâneas, através dos parâmetros genéricos permite a configuração do *baud*, número de *bits* de dados e de paragem, e ainda *bit* de paridade par, impar ou sem *bit* de paridade, não efectua controlo de fluxo e apenas usa os sinais RxD e TxD do conector para a comunicação.

Opções disponíveis a especificar nos parâmetros `generics`:

- `BaudRate`
  1200, 2400, 4800, 9600, 19200, 38400, 115200.
- `DataBits`
  5, 6, 7, 8.
- `Parity`
  'N' (sem paridade), 'E' (paridade par), 'O' (paridade impar).
- `StopBits`
  1 ou 2 (não implementa 1,5).

Os dados a transmitir via série no porto `TxD` devem ser colocados no porto `DataInTx` e o porto `Load` deve ser activado a '1' durante pelo menos um período de *baud*.

Quando são recebidos `DataBits` no porto `RxD`, estes são colocados no porto `DataOutRx` e o porto `RxReady` sinaliza a '1' durante um período de *baud*. No caso de ocorrer um erro de paridade, o porto `RxError` sinaliza a '1' até ao fim da trama, ou seja 2 ou 3 períodos de *baud* conforme o número de *bits* de paragem usado.

Para implementar os diversos *baud rates* sem erros de *jitter* e usar divisores inteiros é necessário ligar um relógio de 72 *MHz* no porto `Clock`.

O sinal de `Reset` é assíncrono activo alto.

Capítulo 4

Utilização por Hardware Template

Sumário

Este capítulo indica a funcionalidade do *hardware template* criado para a fácil utilização e configuração modular das interfaces desenvolvidas, bem como a sua utilização.

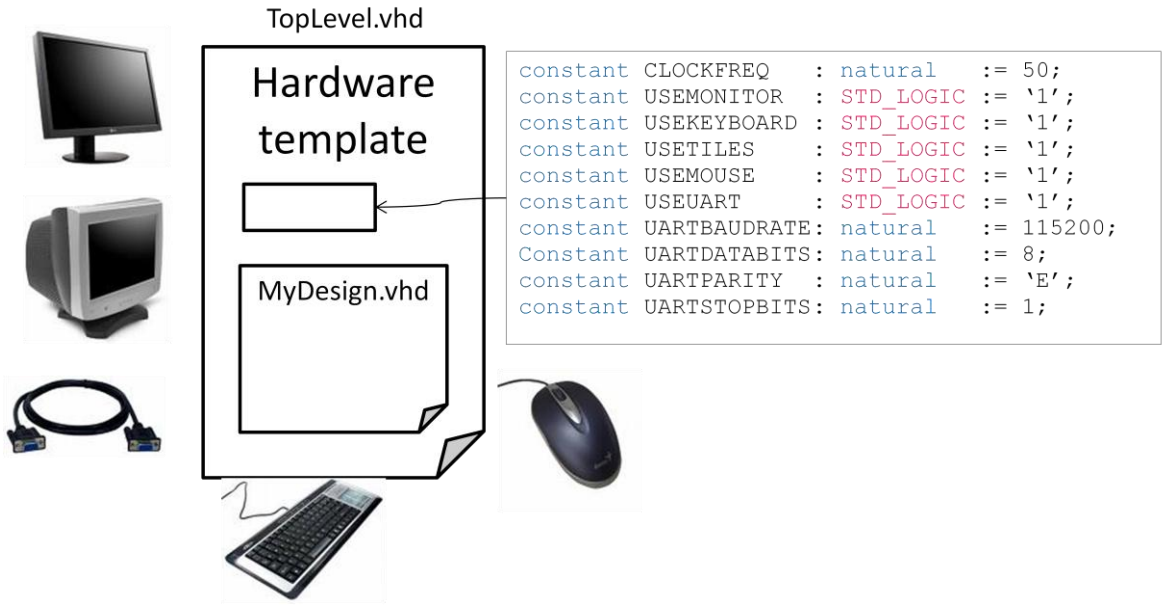
4.1 Introdução

A implementação tradicional requer a instanciação dos módulos pretendidos e definição dos sinais que os interligam, através da abordagem de *hardware template* este passo torna-se mais rápido, eficiente e simples, toda a estrutura está previamente definida e o utilizador apenas tem de inicializar algumas constantes, também elas já pré-definidas.

4.2 Utilização do hardware template

O *hardware template* consiste num ficheiro VHDL da hierarquia mais alta, a declaração da `entity` define os portos ligados ao exterior da FPGA pela ligação aos pinos indicados no UCF (*User Constraints File*).

O ficheiro chama-se portanto TopLevel.vhd e já incorpora as instanciações dos módulos e definições dos sinais que os interligam, é instanciado também um módulo de nome MyDesign.vhd onde apenas a `entity` está correctamente definida segundo as interfaces dos módulos desenvolvidos e o corpo `architecture` se encontra vazio, é lá que o utilizador deve escrever o seu código da aplicação. A selecção dos módulos depende do valor lógico inicializado em algumas constantes cujo nome é elucidativo da sua função, mediante o valor dessas constantes e através de construções `generate` da linguagem VHDL, os circuitos são sintetizados e implementados na FPGA ou não.




```
constant FOREGROUNDCOLOUR : BasicColours := White; -- Black, Blue, Green, Cyan...
constant BACKGROUNDCOLOUR : BasicColours := Blue; -- Black, Blue, Green, Cyan...
```

A selecção não é efectuada com uma constante por módulo mas sim uma constante por funcionalidade, por exemplo, pode desejar-se usar o monitor VGA mas sem suporte de texto, nesse caso os módulos SymbolROM.vhd, RAM.vhd e VGATileMatrix.vhd não são necessários, assim sendo não são instanciados.

As constantes definidas são:

- `constant USEMONITOR : STD_LOGIC := '1';` instancia:
  VGASync.vhd
  RGBMux.vhd

- `constant USEKEYBOARD : STD_LOGIC := '1';` instancia:
  Keyboard.vhd
  (2×Debouncer.vhd)
  (PS2Controller.vhd)
  (KeyboardMapper.vhd)
  (KeyboardText.vhd)

- `constant USETILES : STD_LOGIC := '1';` instancia:
  SymbolROM.vhd
  RAM.vhd
  VGATileMatrix.vhd

- `constant USEMOUSE : STD_LOGIC := '1';` instancia:
  PS2Mouse.vhd
  (2×Devouncer.vhd)
  (PS2Controller.vhd)
  (MouseController.vhd)
  (MouseCursor.vhd)

- `constant USEUART : STD_LOGIC := '1';` instancia:
  UART.vhd

Capítulo 5

Conclusão e trabalho futuro

Sumário

Este capítulo conclui a dissertação com a apresentação de um sumário das contribuições principais resultantes do trabalho desenvolvido.

São também apresentadas algumas propostas para trabalho futuro na continuação deste trabalho de desenvolvimento através de novos núcleos IP e evolução das arquitecturas já implementadas.

5.1 Conclusão

Esta dissertação apresenta o trabalho finalizado que foi desenvolvido no âmbito da proposta apresentada.

Através de várias versões dos módulos desenvolvidos, foram progressivamente sendo atingidos níveis cada vez mais satisfatórios de desempenho e de funcionalidades.

Como se pode verificar, a utilização de recursos é bastante baixa, mesmo para uma FPGA com capacidade média-baixa como a que se encontra na placa NEXIS 2.

	NEXYS 2 XC3S500E-4fg320											
	Flip-Flop's			LUTs			Slices			Block RAMs		
	Usados	Disponíveis	Utilização	Usados	Disponíveis	Utilização	Usados	Disponíveis	Utilização	Usados	Disponíveis	Utilização
Monitor	27	9312	0,29%	55	9312	0,59%	42	4656	0,90%	0	20	0,00%
Monitor + Rato	202	9312	2,17%	439	9312	4,71%	278	4656	5,97%	0	20	0,00%
Monitor + Tiles + Teclado	200	9312	2,15%	1286	9312	**13,81%**	743	4656	15,96%	4	20	20,00%
UART	25	9312	0,27%	40	9312	0,43%	30	4656	0,64%	0	20	0,00%

Uso de memória distribuída para os mapas de bits dos caracteres de texto

Os núcleos IP desenvolvidos sob a forma de vários módulos para utilização de periféricos de interacção com o utilizador, permitem o rápido desenvolvimento de novos sistemas devido à flexibilidade e parametrização proporcionadas bem como o *hardware template* que permite acelerar ainda mais este processo.

É bastante positiva a utilização de métodos visuais através do monitor VGA, quer para apresentação de resultados, interface de utilização gráfica, verificação de valores de *debugging*, demonstração de prova de conceito, etc.

Salienta-se o uso com sucesso de alguns núcleos IP nas aulas de Computação Reconfigurável do ano lectivo transacto, leccionadas pelo Prof. Doutor Valeri Skliarov, foram encontrados ocasionalmente alguns "artefactos" nas imagens do monitor, provavelmente devido a problemas de sincronismo com o restante design desenvolvido pelos alunos, mas sem manifestação de erros de funcionamento.

Destaca-se também a utilização de alguns núcleos IP em trabalhos de dissertação deste mesmo ano:

- "Processador com Conjunto de Instruções Variável Configurável Remotamente"
  João Lima

- "Interacção Remota com Circuitos Implementados em FPGA"
  Abílio Neves

- "Simulação e Teste de Elementos de uma Rede de Tráfego Urbana em FPGA"
  Sérgio Soldado

A criação de um *hardware template* também facilita a utilização dos núcleos IP desenvolvidos, aos mais variados níveis, de ensino, de investigação ou de engenharia.

Os núcleos IP podem ser acedidos através de um *Web site* criado para o efeito, funcionando como um repositório aberto a todas as comunidades académicas, indivíduos amadores ou profissionais. O repositório encontra-se colocado no domínio www.fpga-repository.web.ua.pt disponibilizado pelo CIC-UA [40].

Considera-se portanto que os objectivos propostos para este trabalho foram alcançados, embora como sempre haja espaço para melhorias.

5.2 Trabalho futuro

Dada a natureza desta proposta, uma biblioteca/repositório, parte da premissa duma continuidade de trabalho e constante acumulação de novos módulos com as mais variadas funcionalidades, assim sendo, o trabalho futuro de continuação deste projecto poderá incluir:

- uma análise por outros pontos de vista com o intuito de optimizar, se possível, os módulos já criados
- adição de novos núcleos IP, tais como
  - controlador USB
  - controlador I2C
  - controlador SPI
  - controlador DVI
- melhorias na estrutura do repositório *online*
- adição de *wrappers* com a estrutura de registos adequada para a utilização dos módulos em barramentos de ligação a processadores *soft* ou *hard*.

Capítulo 6

Apêndices

A

Teclado Português e tabela do *Scan-Code Set* 2.

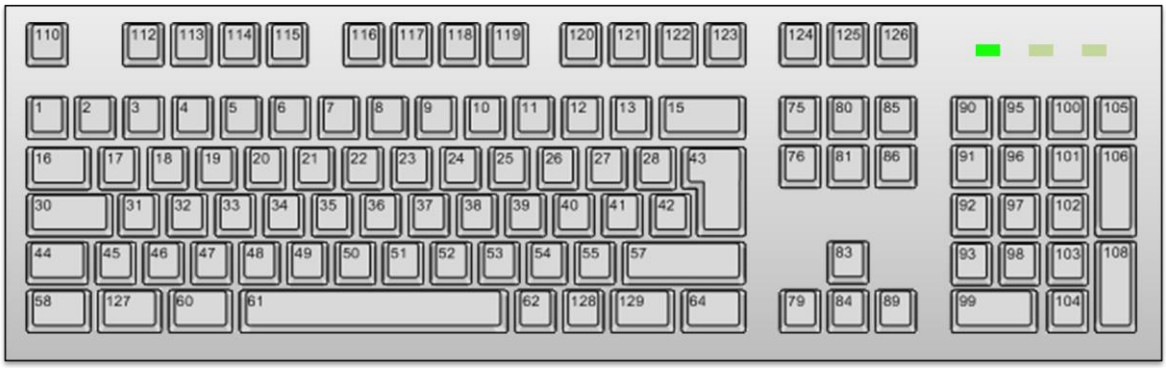


Tabela 6.1 – Códigos do *Scan Code Set* 2.

Tecla #	*Make*	*Break*	Tecla #	*Make*	*Break*	Tecla #	*Make*	*Break*
1	0Eh	F0 0Eh	37	3Bh	F0 3Bh	90	77h	F0 77h
2	16h	F0 16h	38	42h	F0 42h	91	6C	F0 6Ch
3	1Eh	F0 1Eh	39	4Bh	F0 4Bh	92	6Bh	F0 6Bh
4	26h	F0 26h	40	4Ch	F0 4Ch	93	69h	F0 69h
5	25h	F0 25h	41	52h	F0 52h	95	E0 4Ah	E0 F0 4Ah
6	2Eh	F0 2Eh	** 42	5Dh	F0 5Dh	96	75h	F0 75h
7	36h	F0 36h	43	5Ah	F0 5Ah	97	73h	F0 73h
8	3Dh	F0 3Dh	44	12h	F0 12h	98	72h	F0 72h
9	3Eh	F0 3Eh	** 45	61h	F0 61h	99	70h	F0 70h
10	46h	F0 46h	46	1Ah	F0 1Ah	100	7Ch	F0 7Ch
11	45h	F0 45h	47	22h	F0 22h	101	7Dh	F0 7Dh
12	4Eh	F0 4Eh	48	21h	F0 21h	102	74h	F0 74h
13	55h	F0 55h	49	2Ah	F0 2Ah	103	7Ah	F0 7Ah
15	66h	F0 66h	50	32h	F0 32h	104	71h	F0 71h
16	0Dh	F0 0Dh	51	31h	F0 31h	105	7Bh	F0 7Bh
17	15h	F0 15h	52	3Ah	F0 3Ah	106	79h	F0 79h
18	1Dh	F0 1Dh	53	41h	F0 41h	108	E0 5Ah	E0 F0 5Ah
19	24h	F0 24h	54	49h	F0 49h	110	76h	F0 76h
20	2Dh	F0 2Dh	55	4Ah	F0 4Ah	112	05h	F0 05h
21	2Ch	F0 2Ch	57	59h	F0 59h	113	06h	F0 06h
22	35h	F0 35h	58	14h	F0 14h	114	04h	F0 04h
23	3Ch	F0 3Ch	60	11h	F0 11h	115	0Ch	F0 0Ch
24	43h	F0 43h	61	29h	F0 29h	116	03h	F0 03h
25	44h	F0 44h	62	E0 11h	E0 F0 11h	117	0Bh	F0 0Bh
26	4Dh	F0 4Dh	64	E0 14h	E0 F0 14h	118	83h	F0 83h
27	54h	F0 54h	75	E0 70h	E0 F0 70h	119	0Ah	F0 0Ah
28	5Bh	F0 5Bh	76	E0 71h	E0 F0 71h	120	01h	F0 01h
* 29	5Dh	F0 5Dh	79	E0 6Bh	E0 F0 6Bh	121	09h	F0 09h
30	58h	F0 58h	80	E0 6Ch	E0 F0 6Ch	122	78h	F0 78h
31	1Ch	F0 1Ch	81	E0 69h	E0 F0 69h	123	07h	F0 07h
32	1Bh	F0 1Bh	83	E0 75h	E0 F0 75h	125	7Eh	F0 7Eh
33	23h	F0 23h	84	E0 72h	E0 F0 72h	127	E0 1Fh	E0 F0 1Fh
34	2Bh	F0 2Bh	85	E0 7Dh	E0 F0 7Dh	128	E0 27h	E0 F0 27h
35	34h	F0 34h	86	E0 7Ah	E0 F0 7Ah	129	E0 2Fh	E0 F0 2Fh
36	33h	F0 33h	89	E0 74h	E0 F0 74h			

Tecla #	*Make*	*Break*
124	E0 12 E0 7Ch	E0 F0 7C E0 F0 12h
126	E1 14 77 E1 F0 14 F0 77h	não existe

* apenas existe nos teclados baseados no modelo de 101 teclas (tipo teclado USA)

** apenas existe nos teclados baseados no modelo de 102 teclas (tipo teclado Português)

B

Lista de Acrónimos

A/N	*Alphanumeric*
AMBA	*Advanced Microcontroller Bus Architecture*
APA	*All Points Addressable*
ARM	*Advanced RISC Machines, Ltd.*
ASCII	*American Standard Code for Information Interchange*
ASIC	*Application-Specific Integrated Circuit*
AT	*Advanced Technologies*
BAT	*Basic Assurance Test*
CAD	*Computer-Aided Design*
CGA	*Colour Graphics Adapter*
CIC-UA	Centro de Informática e Comunicações da Universidade de Aveiro
CLB	*Configurable Logic Block*
CLK	*Clock*
CMOS	*Complementary Metal-Oxide Semiconductor*
CPLD	*Complex Programmable Logic Device*
CPU	*Central Processing Unit*
DCE	*Data Circuit-terminating Equipment*
DCM	*Digital Clock Manager*
DDC	*Display Data Channel*
DDR	*Double Data Rate*
DETI	Departamento de Electrónica, Telecomunicações e Informática
DIN	*Deutsches Institut für Normung e.V.*
DSP	*Digital Signal Processing*
DTE	*Data Terminal Equipment*
DVI	*Digital Visual Interface*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*

EGA	*Enhanced Graphics Adapter*
FIFO	*First-In First-Out*
FPGA	*Field-Programmable Gate Array*
FSM	*Finite State Machine*
GND	*Ground*
GTF	*Generalized Timing Formula*
HDL	*Hardware Description Language*
I2C	*Inter-Integrated Circuit*
IEEE	*Institute of Electrical and Electronics Engineers*
IOB	*Input/Output Block*
IP	*Intellectual Property*
ISE	*Integrated Software Environment*
JTAG	*Joint Test Action Group*
LAB	*Logic Array Block*
LCD	*Liquid Crystal Display*
LED	*Light Emitting Diode*
LUT	*Look-Up Table*
MAC	*Multiply and Accumulate*
MSI	*Medium-Scale Integration*
NoC	*Network-on-Chip*
NRE	*Non-Recurring Engineering*
PC	*Personal Computer*
PCB	*Printed Circuit Board*
PCI-E	*Peripheral Component Interconnect – Express*
PLD	*Programmable Logic Device*
PS/2	*Personal System 2*
RAM	*Random Access Memory*
RAMDAC	*Random Access Memory and Digital-to-Analog Converter*
RC	Resistivo-Capacitivo
RGB	*Red Green Blue*
RISC	*Reduced Instruction Set Computer*
ROM	*Read Only Memory*
RS	*Recommended Standard*
SATA	*Serial Advanced Technology Attachment*
SoC	*System on Chip*
SPI	*Serial Peripheral Interface Bus*
SRAM	*Static Random Access Memory*
SSI	*Small-Scale Integration*
TIA	*Telecommunications Industry Association*

TTL	*Transistor-Transistor Logic*
UART	*Universal Asynchronous Receiver-Transmitter*
UCF	*User Constraints File*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*
VESA	*Video Electronics Standards Association*
VGA	*Video Graphics Array*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*
VLSI	*Very Large-Scale Integration*
XT	*Extended Technology*

Capítulo 7

Bibliografia

[1] Gordon Moore, "Cramming More Components onto Integrated Circuits," *Electronics Magazine*, vol. 38, no. 8, April 1965.

[2] Xilinx Inc. [Online]. http://investor.xilinx.com/phoenix.zhtml?c=75919&p=irol-irhome

[3] Xilinx Inc. , Spartan-3 FPGA Family Data Sheet. [Online]. http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[4] Actel. Introduction to Actel FPGA architecture. [Online]. http://www.actel.com/documents/Actel_Architecture_AN.pdf

[5] Stephen Brown and Jonathan Rose, "Architecture of FPGAs and CPLDs: A Tutorial," University of Toronto,.

[6] Altera Corp. The Stratix II Logic and Routing Architecture. [Online]. http://www.altera.com/literature/cp/cp-01005.pdf

[7] Xilinx Inc. , Spartan-3 Generation FPGA User Guide. [Online]. http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[8] Clive Maxfield, *The Design Warrior's Guide to FPGAs - Devices, Tools and Flows*.: Newnes, 2004, ISBN: 0-7506-7604-3.

[9] IBM Semiconductor Solutions. IBM Power Architecture. [Online]. http://www-03.ibm.com/technology/power/licensing/coreconnect/index.html

[10] OpenCores. OpenCores Wishbone bus. [Online]. http://www.opencores.org/projects.cgi/web/wishbone/wishbone

[11] Advanced RISC Machines, Ltd, AMBA - Advanced Microcontroller Bus Architecture Specification, 1997.

[12] Advanced RISC Machines, Ltd. AMBA Overview. [Online]. http://www.arm.com/products/solutions/AMBAHomePage.html

[13] Peter Wilson, *Design Recipes for FPGAs*.: Newnes, 2007, ISBN: 978-0-7506-6845-3.

[14] Pong P. Chu, *FPGA Prototyping by VHDL Examples - Xilinx Spartan-3 Version*.: John Wiley & Sons, Inc., 2008, ISBN: 978-0-470-18531-5.

[15] Stephen A. Edwards, "Experiences Teaching an FPGA-based Embedded Systems Class," in *Proceedings of the workshop on embedded systems education (WESE)*, Jersey City, New Jersey, USA, 2005.

[16] Valery Sklyarov, Iouliia Skliarova, Bruno Pimentel, and Manuel Almeida, "Remote Reconfigurable Systems for Electronic Engineering and Education," in *Proceedings of the 2nd International Conference on Communications and Electronics – HUT-ICCE'2008*, Hoi An, Vietnam, 2008, pp. 394-399.

[17] James O. Hamblen, "Using Large CPLDs and FPGAs for Prototyping and VGA Video Display Generation in Computer Architecture Design Laboratories," in *4th Annual Workshop on Computer Architecture Education (WCAE-4)* , Las Vegas, Nevada, USA, 1998.

[18] Manuel Almeida, Valery Sklyarov, Iouliia Skliarova, and Bruno Pimentel, "Design Tools for Reconfigurable Embedded Systems," in *Proceedings of the 2nd International Conference on Embedded Software and Systems*, China, 2005, pp. 254-261.

[19] Iouliia Sliarova, "Desenvolvimento de circuitos reconfiguráveis que interagem com um monitor VGA," *Revista do DETUA*, vol. 4, no. 5, pp. 626-631, Setembro 2005.

[20] Project Supervisor: Peter Sutton. VHDL XSV Board Interface Projects. [Online]. http://www.itee.uq.edu.au/~peters/xsvboard/index.html

[21] OpenCores.org. [Online]. http://www.opencores.org

[22] ALSE - Advanced Logic Synthesis for Electronics. [Online]. http://www.alse-fr.com/English/ips.html

[23] Cooperative Analog/Digital Signal Processing Laboratory at Georgia Tech. Xilinx Resource Page. [Online]. http://www.ece.gatech.edu/academic/courses/fpga/Xilinx/

[24] Digilent Inc. Digilent Nexy2 Board Reference Manual. [Online]. http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf

[25] Digilent Inc. [Online]. http://www.digilentinc.com

[26] Jack G. Ganssle. (2008) A Guide to Debouncing. [Online]. http://www.ganssle.com/debouncing.pdf

[27] Altera Corp., "Understanding Metastability in FPGAs," White paper 2009.

[28] Texas Instruments, "Metastable Response In 5-V Logic Circuits," Application Note 1997.

[29] Adam Chapweske. Computer-Engineering - PS/2 Protocol. [Online]. http://www.computer-engineering.org/ps2protocol

[30] IBM, Keyboard and auxiliary device controller reference manual, 1990.

[31] IBM. IBM archives:1984. [Online]. http://www-03.ibm.com/ibm/history/history/year_1984.html

[32] IBM. IBM archives:1987. [Online]. http://www-03.ibm.com/ibm/history/history/year_1987.html

[33] Adam Chapweske. Computer-Engineering - PS/2 Keyboard. [Online]. http://www.computer-engineering.org/ps2keyboard/

[34] Microsoft, Keyboard Scan Code Specification.

[35] IBM, Video subsystem reference manual, 1990.

[36] VESA. Video Electronics Standards Association. [Online]. www.vesa.org

[37] VESA. Generalized Timing Formula. [Online]. http://www.vesa.org/Public/GTF/GTF_V1R1.xls

[38] Electronics Industries Association, EIA Standard RS-232-C Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Data Interchange, 1969.

[39] Telecommunications Industry Association, 9-Position Non-Synchronous Interface between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange, 1990.

[40] Centro de Informática e Comunicações da Universidade de Aveiro. Serviços web.ua. [Online]. http://www.ua.pt/cic/PageText.aspx?id=1884

[41] Adam Chapweske. Computer-Engineering - PS/2 Mouse. [Online]. http://www.computer-engineering.org/ps2mouse/